



# 探寻数据背后的逻辑

## R语言数据挖掘之道

宋云生 张坚洪 黎新年◎著

电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING

## 内 容 简 介

数据分析、数据挖掘的本质是探寻数据背后的逻辑,挖掘人们的欲望、需求、态度等。本书不仅仅教会读者如何掌握数据挖掘相关技能,更教会读者如何从数据挖掘结果中分析出更深层次的逻辑。

本书主要介绍使用 R 语言进行数据挖掘的过程。具体内容包括 R 软件的安装及 R 语言基础知识、数据探索、数据可视化、回归预测分析、时间序列分析、算法选择流程及十大算法介绍、数据抓取、社交网络关系分析、情感分析、话题模型、推荐系统,以及数据挖掘在生物信息学中的应用。另外,本书还介绍了 R 脚本优化相关内容,使读者的数据挖掘技能更上一层楼。

本书适合从事数据挖掘、数据分析、市场研究的工作者及学生群体,以及对数据挖掘和数据分析感兴趣的初级读者。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

## 图书在版编目(CIP)数据

探寻数据背后的逻辑: R 语言数据挖掘之道 / 宋云生, 张坚洪, 黎新年著. —北京: 电子工业出版社, 2018.8  
ISBN 978-7-121-33861-8

I. ①探… II. ①宋… ②张… ③黎… III. ①数据采集—研究 IV. ①TP274

中国版本图书馆 CIP 数据核字(2018)第 049541 号

策划编辑: 王 静

责任编辑: 石 倩

印 刷: 三河市双峰印刷装订有限公司

装 订: 三河市双峰印刷装订有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

开 本: 787×1092 1/16 印张: 27 字数: 742.4 千字

版 次: 2018 年 8 月第 1 版

印 次: 2018 年 8 月第 1 次印刷

定 价: 89.00 元

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888, 88258888。

质量投诉请发邮件至 [zltz@phei.com.cn](mailto:zltz@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式: 010-51260888-819, [faq@phei.com.cn](mailto:faq@phei.com.cn)。

# 序言

## 提纲挈领式心诀： 一名数据挖掘工程师的成长之路

### 我的学习之路

不知不觉毕业两年多了，有一些大音如霜工作室的读者总想了解一下我是怎么学习数据挖掘、数据分析的，下面就综合大家常见的问题分享一下自己的经历、经验。

我不是学数学的，也不是学计算机的，研究生的专业是植物学，而且方向是植物分类，可以说很难和数据挖掘、市场研究等领域扯上关系。唯一能扯上关系的也就是我的舍友是做生物信息学研究的。

说一句丢人的话，在读本科时上的 SPSS 课我都不知道在讲什么。那时没考过计算机等级考试，原因是我每分钟打汉字的速度都不过关，讲这么多，只是为了告诉读者，我的基础并不扎实。

需要说明的是，我的英语还不错，在大一和大二分别通过了英语四、六级考试（而其他科目则学得比较一般，因为我每学期只有一两个主要学习目标），在大四我读了很多英文文献。因为在读研究生期间需要查阅大量文献，我需要给这些文章建立一个数据库，于是年少无知的我就选择了 Access。选择 Access 的原因并不是我比较熟悉它，而是我的老师用它，我至今也不会太多的操作。这应该算是我开始接触数据分析了。

使用高级语言时，记不住函数不要紧，但是你要有很强的搜索能力。

之所以讲这一段经历，不是为了说明我起步晚，而是为了说明建立 Access 文献数据库锻炼了我的英文搜索能力。我一碰到问题，就在 Google 里搜索，很快就能找到答案。于是 Google 几乎成了我的眼睛，真正做到了用 Google 搜索、发邮件、社交、阅读和写作。在公司里曾经传说，如果是连我都搜不到的内容，那么别人更不可能搜到。有些年轻人就怕英文，我并不是崇洋媚外，客观地想一想，现在的很多知识都是从欧美起源的，如果你连这门世界语言都不掌握，那么你获得的资料永远都是二手资料。另外，无论你是找函数还是找包、模块，抑或是为问题寻找答案，使用 Google 进

## 探寻数据背后的逻辑：R 语言数据挖掘之道

行英文搜索会为你节省很多时间。掌握这门语言并不需要你听、说、读、写样样精通，而是将其作为一种工具，应用起来比较方便就可以了。

要善用英文搜索，原因很简单，你所用的编程语言或软件大多是外国人构建的，并且在国外已经普及，相关的问答社区早已完善，你碰到的问题可能早就有人解决了。

在搜索文献的过程中，我喜欢上了《经济人》的 *Graphic Details* 栏目，发现其绘制的图表非常漂亮、专业，于是我就开始学习 Excel，尽自己所能将 Excel 图表做得更漂亮、更专业，这些经历为我日后做数据可视化打下了坚实的基础：我知道了商务色彩搭配及图表要简洁、易读等原则，我知道怎么使自己的图表特色鲜明。后来看了大前研一先生的著作，了解了专业精神，我曾经写下这样一句话，以勉励自己：

所谓专业，即每一个细节都经得起推敲。

有一天，舍友看到我用 Excel 作图，嘲笑我孤陋寡闻，推荐我学习 R 语言，然后我就开始搜寻一些 R 语言入门读物进行阅读，慢慢地知道了关于这门语言的粗浅知识。

这个时候已经到研二下半学期了，我需要为自己未来的工作做打算了：是步入园林行业还是就此转行？必须做一个决断。我发现自己真的对植物分类不感兴趣，而我做家教的学生的妈妈是星空传媒的一个经理，平时待我很好，她说毕业可以介绍我去做市场研究。我了解了一下市场研究，发现其中涉及一些数据分析的内容（现在看起来很简单），于是，我从此决定踏上数据分析这条“不归路”。

为了快速上手、熟悉统计学知识，我并没有马上深入地学习 R 语言，而是像以往一样懒懒散散地学习（后悔当时没有实战学习）。我通过搜索发现，市场研究的岗位大多将熟练使用 SPSS 作为硬性要求，偶尔也会要求熟悉 R 语言，但 SPSS 对我来说更容易上手，于是就开始学习 SPSS。SPSS 帮助我巩固了统计学知识，当学习完简单的统计学知识后，我发现 SPSS 不够灵活，很多功能不够用，做出的图表很难看（这对于我来说是无法忍受的），因此，网络上有一些人鄙视 SPSS，但很推崇 R 语言。于是我决定要深入地学习 R 语言。我先将 SPSS 的功能在 R 中做了一遍，有了一些自己的理解后，我开始在自己的论文里做一些数据分析的内容。

现在想来，如果我直接在实战中学习可能会节省更多的时间。

实战更能锻炼技能水平。

研二快结束了，开始找工作了。我找工作的目的很明确，如果工作不是做数据分析、数据研究的，那么我宁愿放弃这个工作的机会。非数据研究的岗位我也不去面试，这样又省下了大量的时间学习。

## 在工作中学习

2013 年毕业后，我去了一家医药市场研究公司，当时的工作并不太忙，我有大量的时间学习。但这时也暴露了我的弱点，公司的数据并不是很规整（raw data）的，往往需要标准化等，而且数据规模也不再是之前练习时那么小，在面对这些脏数据、大一点的数据时，我的数据清洗水平显得捉襟见肘。周围的人都是 Excel 高手，如果跟着他们学，估计也能成为高手，但是我一定要在 R 中做数据清洗整理，反正公司的工作不是很多，我就一点点地学习和积累，这样我的数据处理能力就逐渐扎实起来了。其间我用两天读完了《异类》这本书，感触很深，阅读经历已经写成一篇文章在我们的公众号里分享了。

任何一个工具在刚开始学习时都会觉得它很糟糕，其实这并不是工具的问题，而是自己的知识体系跟不上节奏，或者是它的很多方法与自己原有的认知相反，这时不要急于否定它，而是要深入地学习它。知识体系是一个积累过程，为自己准备一万个小时计划吧。

我们公司当时在做 BI（商业智能），于是我接触了市面上常见的 BI 工具，包括 Tableau、QV 等，我熟悉它们的优、劣势，也熟悉它们的数据可视化效果。因为需要将 R 语言的页面融入 BI 中，所以我熟悉了 shiny 包，做了一些页面，但我渐渐看到 R 语言在做这些通用语言的工作时所暴露的缺点，于是开始接触 Python。

后来，我们的合作公司的总经理听说我比较熟悉 R 语言，就向我请教，我们一起讨论了 R 语言和数据挖掘。得知他们在做文本挖掘，于是在我闲暇时间开始学习中文文本挖掘的内容。没有成型的数据和书，我就看帖子，去一个个地实现，然后积累经验，这时我对 R 的操作算得上非常熟练了，从实现到速度优化（并行计算等）等也已经非常熟练，积累的代码也非常多了。

后来，那个经理找我做医院处方数据挖掘工作，之后，他请我去负责法院文本数据挖掘，我没去，但成了他们的外援，仍然没收过钱，他们搭建的一台服务器也帮助我了解了不少 Linux 的知识。

刚开始，锻炼自己的机会远远比钱重要，反正自己闲着也是闲着，但是这种情况只适用于刚开始。

后来，我们公司推出了微信公众号平台，我开始给公司的公众号写文章。其间我为公司的公众号写了多篇关于综合排名的文章，阅读量最高达到 4 万多人次，当时公众号的粉丝才 2 000 人左右。后来我又制作了评价医院市场趋势的综合指标体系，现在公司也一直在沿用这套指标，这些工作中的小点子都是我在公交车上想出来的。

除要把工作当成一种谋生手段外，还必须有极大的兴趣，要么不做，要么做好。

另外，我在公交车上读完了 *Data Mining with R learning by case studies*、*Machine Learning for Hackers*、*R Graphics Cookbook* 等书籍，之所以提这三本书，是因为我不止读过一遍，这三本书很有特色，第一本帮助我学习了各种算法，第二本帮助我接触了实际应用中的知识，第三本帮我熟练了 ggplot 的函数及图表元素结构。我开始学会利用零散的时间，坚持积累，也开始学习高度自律。

## 探寻数据背后的逻辑：R 语言数据挖掘之道

古之成大事者，不唯有超世之才，亦必有坚韧不拔之志。

——苏轼

其实，我一直幻想着有一个自己想写什么就写什么的平台，于是，我和小伙伴们开通了微信公众号，直到现在，我们更注重文章的可读性、趣味性，而不仅仅是技术，但是每一篇文章都可以作为一个小项目让希望学习数据分析的读者能锻炼一下自己的技能。

经常有读者问学数据分析就一定要学编程吗？以及为什么要看英文资料？针对这两个问题，我写下了这样一段对话，希望你能在对话中找到答案。

为什么学习数据分析？

赚钱！

什么样的人容易赚钱？

技能比别人高的！

英语是不是一般人的难关？是不是大家都想学习傻瓜式操作软件？

是！

那么如果大家都这么想，你应该怎么做？

很明显，你要做其他人不愿意做的事情，才能赚到别人不能赚的钱！

作为数据分析师，一定要将自己和技术区分开，分析数据、挖掘数据本质上是探寻数据背后的人心，挖掘人们的欲望、需求、态度等，所以数据分析师还要尽量拓宽自己的视野和知识结构，尽自己所能博览群书。

我的经历大体如此，中间会有各种迷茫、各种苍白无力，但是如果你缺少什么，就去搜集资料，做出判断，努力去争取，这一点总不会错，千万不要一味地否定你不了解的东西，这也是我对待未知领域的态度。

作为一名技术人员，要让自己的知识时刻在进步！这是一种宿命。

作 者

# 前言

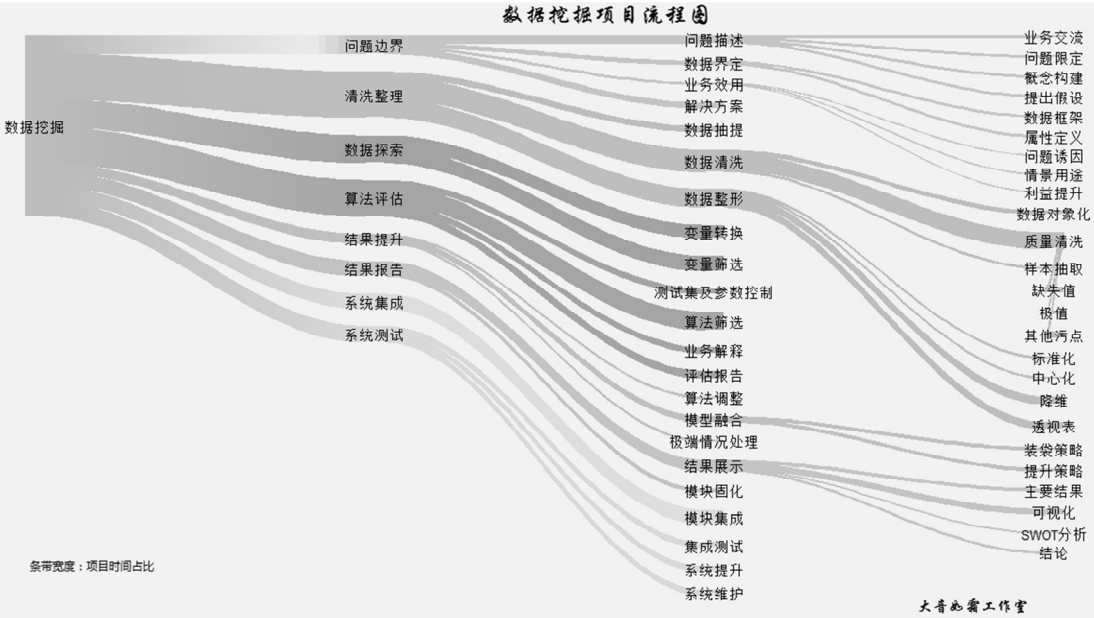
## 什么是规范化的数据挖掘流程

人总是被自己日常从事的工作所蒙蔽双眼，看不到事态发展的整体面貌，为了手里的工作而工作，这就是所谓的迷失吧。一个数据挖掘项目不仅仅是数据挖掘工程师手中的一部分工作（虽然它是工作的核心），作为一个力求向上的人，要跳出来看看项目的全貌。只有对项目全程有了足够的了解，才能更加有效地使用数据挖掘、机器学习、数据分析的工具。

数据挖掘项目一般可以分为问题边界、清洗整理、数据探索、算法评估、结果提升、结果报告、系统集成和系统测试 8 个主要的模块。一般而言，可以尽量将这些模块合并，但无论怎么合并，它们在项目中都是不可或缺的（当然，有些项目并没有其中某些模块的需求，比如市场研究项目，它们可能就不需要系统集成）。从右图中可以看到数据清洗、数据探索、算法评估占据了项目的大部分时间，这也说明它们是项目的核心内容，缺了这 3 项，就不能再称为是数据挖掘项目了。

项目	项目分支	时间占比%
数据挖掘	问题边界	10
数据挖掘	数据清洗	25
数据挖掘	数据探索	15
数据挖掘	算法评估	20
数据挖掘	结果提升	5
数据挖掘	结果报告	8
数据挖掘	系统集成	8
数据挖掘	系统测试	9

数据挖掘项目流程如下图所示。



### 问题边界

问题边界一般是项目的开头部分，可以分为 4 点。首先要和业务部门细致沟通，从业务背景中提炼出对业务问题的描述，限定项目要解决的问题，便于组织力量集中对这些问题设计解决方案。然后根据解决构想将业务问题转换为数据语言，限定将要使用的数据界限，搞清楚要牵涉哪些数据。之后为了吸引业务部门必须整理出业务效用，告诉业务部门如果解决这些问题能够得到哪些改善，完成业务部门哪些具体目标。最后要将以上问题整理成一个可行的解决方案。很多人忽略了这个阶段，其后果就是业务部门觉得挖掘出来的结果不是他们想要的，或者节外生枝补充各种相关的或不相关的业务问题，最终扭曲了项目本身，使工作反复无常。所以，在项目实施之前，非常仔细地沟通并制定一个完善的问题边界非常重要。

### 清洗整理

清洗整理是数据挖掘工程师非常熟悉的工作，但是，很少有人认识到这是项目中花费时间最多的部分，很多人会以为算法评估部分才会花费最多的时间。其实不然，如果数据清洗进行得不顺利，则将直接影响后面的工作和模型的效果。首先要设计畅通、高效的数据抽取程序，将数据从各种数据平台抽取出来供数据挖掘工具使用，然后进行数据清洗，将数据转化为数据挖掘工具便于处理的对象类型（在 R 里指 list、data.frame、array 等），再进行质量清洗，包括处理缺失值、异常值、其他污点（在文本挖掘中多如牛毛）等。之后要对数据整形，包括一些统计变化，例如中心化、标准化、降维等，更重要的是数据形状的变化。

还有一项就是数据抽样，面对大数据，在数据处理阶段就要进行抽样，不能因为要清洗一个点就清洗全量的数据，那样会花费大量的时间。不如抽取小样本进行测试，等进行完数据清洗程序后，再进行全量数据的整体清理，这样反而更加省时省事，这里的样本量需要尽量保证抽到足够多的问题数据，同时要让程序运行起来非常轻松、高效。

### 数据探索

数据探索要完成两个目标：变量转换和变量筛选。其中变量转换既包括变量的重新计算变形，也包括概念变量的构建，比如，在客户流失预警项目中要定义什么样的客户是流失客户，就会产生出一个新的变量。如果这个变量的定义不能用业务进行合理解释，那么下面的工作就是“瞎子点灯白费油”了。有些变量不仅不会对模型产生正向的影响，而且除了影响速度，还会降低模型的效果。显然进行变量筛选就非常重要了。谷歌预测流感模型筛选变量足以证明数据探索多么重要，而且在大数据环境下，数据探索已不再是一件轻而易举就能完成的事情了。

### 算法评估

算法评估是数据挖掘项目的灵魂。算法评估首先要求我们充分了解算法或模型的参数意义，然后需要预留测试数据集。模型评估不是仅仅比较模型结果的准确性是否存在差异（别忘了统计学教导我们比较差异时要判断差异的显著水平），所以，模型比较是对不同模型准确性均值的比较。算法筛选完成后，工作就告一段落了，这时要和业务部门一起对结果进行业务解释，不能进行业务解释



的数据挖掘结果就是为了数据挖掘而数据挖掘，这显然就是迷失在了项目中，遗忘了项目要解决的问题边界。最后要对结果进行完整的评估报告。评估报告是必需的，因为除了将它给领导看，更重要的是它能帮助你总结发现这个过程中可以改善的节点。

### 结果提升

首先要判断是否需要调整算法或模型，包括更换算法或调整参数。如果模型调整没有必要，那么就要考虑使用模型融合提高模型效率。模型融合的方法包括装袋（Bagging）和提升（Boosting）等，有些方式可能用业务解释起来比较困难，这也是数据挖掘工程师要考虑的问题之一，显然，有些问题选择可解释模型比较好。在项目中对一些极端情况最好另做处理。

### 结果报告

“丑媳妇也要见公婆，”分析结果报告最终要给业务部门的同事学习，教他们如何使用数据挖掘的结果进行业务分析和部署，其中主要成果要突出，吸引他们的眼球，一定要联系业务具体的困境或具体的业务情景，即所谓的对症下药。规律和结果必须通过易读的方式传达给受众，充满技巧的数据可视化是不二之选，将美妙的可视化图表嵌入具体的应用情景中进行宣讲，往往能达到事半功倍的效果，因此，在此处无论多么努力都不为过。SWOT 分析是业务部门最喜欢的分析方式，我们当然不能放过，以对方熟悉的方式表达自己的诉求，是交流的法宝。

模块固化这一步工作的快慢取决于之前的工作，如果之前已经考虑到后面要进行模块固化，那么就会将代码写得比较规范、注释良好，这种情况下就很容易将数据清洗、数据整形、变量转换、模型构建、结果输出等模块的内容固定下来，成为一个数据有进口及有出口的脚本文件。

### 系统集成

将固化下来的模块按照一定的秩序集成在一起，就成为一个分析的脚本体系。在这个体系中，有输入就有产出，中间不需要人工干预，是一个有序的自动化脚本体系。这一步考验数据挖掘工程师对每一步任务的理解。良好的模块集成可以提升整个系统的速度，减少后期维护的时间和次数。

模块集成后要与其他系统集成在一起，首先要和数据平台（数据库、Hive、Hadoop）对接，为分析模块提供数据来源和存储分析结果，同时要和前台展示对接，将结果可视化，让结果真正接触受众，即所谓的为决策者提供支持。

### 系统测试

这么一个“五脏俱全”的系统需要维护在所难免，总有一些极端情况会导致数据分析模块宕机，所以，代码一定要写得尽量规范，注释要尽量清晰，否则在维护时会有一种再造系统的感觉。关于规范请参看 *Google's R Style Guide*。

# 目录

第 1 章 万事不只开头难 .....	1
1.1 工欲善其事，必先利其器：安装 .....	1
1.1.1 安装 R 和 RStudio .....	1
1.1.2 安装数据包 .....	3
1.1.3 数据包加载、卸载、升级，查看帮助文档 .....	5
1.1.4 什么样的 R 包值得相信 .....	7
1.2 了解 R 的对象 .....	8
1.2.1 如何进行常见的算术运算 .....	8
1.2.2 R 语言的三大数据类型 .....	10
1.2.3 向量及其运算 .....	12
1.2.4 因子变量鲜有人知的秘密 .....	15
1.2.5 矩阵相关运算及神奇的特征值 .....	17
1.2.6 数据框及其筛选、替换、添加、排序、去重 .....	18
1.2.7 与数组（array）相比，表单（list）的用处更加广泛 .....	22
1.2.8 如何进行数据结构之间的转化 .....	23
1.3 R 语言的重器：函数 .....	26
1.3.1 自编函数 .....	26
1.3.2 有用的 R 字符串函数 .....	29
1.4 控制流在 R 语言里只是一种辅助工具 .....	31
1.4.1 判断 .....	32
1.4.2 循环 .....	33
1.5 数据的读入与输出 .....	35
1.5.1 常见数据格式的输入 / 输出（CSV、TXT、RDATA、XLSX） .....	35
1.5.2 数据库连接：Oracle、MySQL 及 Hive .....	37
1.5.3 乱码就像马赛克一样让人讨厌 .....	39

第 2 章 数据探索，招招都是利器 .....	41
2.1 不要在工作后才认识“脏数据” .....	41
2.1.1 以老板信服的方式处理缺失数据 .....	42
2.1.2 异常值预警 .....	48
2.1.3 字符处理正则表达式不再是天书 .....	49
2.2 数据透视、数据整形、关联融合与批量处理 .....	50
2.2.1 还忘不掉 Excel 的数据透视表吗 .....	50
2.2.2 你能给数据做整形手术吗：long 型和 wide 型 .....	52
2.2.3 关联合并表 .....	54
2.2.4 数据批处理：R 语言里最重要的一个函数家族：*ply .....	55
2.3 一招完成数据探索报告 .....	58
2.4 拯救你的很多时候是基础理论 .....	61
2.4.1 参数检验及非参检验 .....	62
2.4.2 学了很多算法却忘了方差分析 .....	68
2.4.3 多因素方差分析及协方差作用 .....	70
2.4.4 很多熟悉的数据处理方法已经成笑话，工具箱该换了 .....	73
第 3 章 从商务气质的数据可视化说起 .....	84
3.1 说说数据可视化的专业素养 .....	84
3.1.1 数据可视化历史上有多少背影等你仰望 .....	84
3.1.2 商务图表应该具有哪些素质 .....	87
3.1.3 那些你不知道的图表误导性伎俩 .....	94
3.1.4 如何快速解构著名杂志的图表 .....	98
3.2 ggplot2 包：一个价值 8 万美元的态度 .....	103
3.2.1 一张图学会 ggplot2 包的绘图原理 .....	105
3.2.2 基础绘图科学：ggplot2 包的主题函数继承关系图（关系网络图） .....	127
3.2.3 基础图表一网打尽 .....	132
3.2.4 古老的地图焕发新颜 .....	151
3.3 将静态图转为 D3 交互图表：plotly .....	156
3.4 从基础到进阶的变形图表 .....	157
3.4.1 马赛克图（分类变量描述性分析） .....	157
3.4.2 Sankey 图和 chordDiagram 图 .....	158
第 4 章 分位数回归模拟股票指数风险通道 .....	163
4.1 用线性回归预测医院的药品销售额 .....	163

4.2	多项式回归及常见回归方程的书写 .....	168
4.3	Lasso 回归和回归评价的常见指标 .....	170
4.4	分位数回归拟合上证指数风险通道 .....	175
第 5 章	时间序列分析 .....	181
5.1	时间序列分析：分析带有时间属性的数列 .....	181
5.2	不是所有序列都叫时间序列 .....	181
5.3	时间序列三件宝：趋势、周期、随机波动 .....	183
5.3.1	趋势 .....	183
5.3.2	周期 .....	184
5.3.3	随机波动 .....	186
5.4	预测分析 .....	186
5.4.1	指数平滑法 .....	186
5.4.2	ARIMA 模型预测 .....	188
第 6 章	选择什么算法也有一套流程 .....	192
6.1	重新审视一下这几个模型 .....	192
6.1.1	Logistic 回归 .....	192
6.1.2	我要的不是一棵树，而是整座森林：随机森林 .....	195
6.1.3	神奇的神经网络 .....	196
6.2	银行信用卡评估模型之变量筛选 .....	197
6.2.1	变量构建 .....	197
6.2.2	Logistic 回归变量筛选 .....	198
6.2.3	随机森林变量筛选 .....	203
6.2.4	人工神经网络建模 .....	204
6.3	必须面对的模型评估 .....	204
第 7 章	深入浅出十大算法 .....	208
7.1	C5.0 算法 .....	208
7.1.1	一个重要的概念：信息熵 .....	208
7.1.2	非列变量选择的实例 .....	209
7.1.3	C5.0 算法的 R 实现 .....	210
7.2	K-means 算法 .....	212
7.2.1	K-means 算法的 R 实现 .....	212
7.2.2	怎么确定聚类数 .....	213

7.3	支持向量机 (SVM) 算法.....	213
7.3.1	通俗理解 SVM .....	214
7.3.2	SVM 的 R 实现.....	216
7.4	Apriori 算法 .....	216
7.4.1	举例说明 Apriori .....	217
7.4.2	Apriori 算法的 R 实现.....	219
7.5	EM 算法 .....	220
7.5.1	举例说明 EM 算法 .....	221
7.5.2	EM 算法的 R 实现 .....	222
7.6	PageRank 算法.....	223
7.7	AdaBoost 算法 .....	224
7.8	KNN 算法与 K-means 算法有什么不同 .....	226
7.9	Naive Bayes (朴素贝叶斯) 算法 .....	227
7.10	CART 算法.....	228
第 8 章	数据抓取.....	231
8.1	数据挖掘工程师不可抱怨“巧妇难为无米之炊” .....	231
8.2	抓取股市龙虎榜数据, 碰碰运气 .....	232
8.2.1	了解 XML 和 Html 树状结构, 才能庖丁解牛 .....	233
8.2.2	了解 R Curl 包和网页解析函数 .....	234
8.2.3	抓取股票龙虎榜 .....	235
8.2.4	资金流入分析 .....	237
8.3	抓取某家医药信息网站全站药品销售数据 .....	240
8.3.1	所有医药公司名称一网打尽 .....	240
8.3.2	为什么抓取数据时可以使用 For 循环.....	242
8.3.3	不要把代码写复杂 .....	244
8.3.4	用 Sankey 数据流描绘医药市场份额流动.....	248
第 9 章	不可不说的社交网络关系 .....	254
9.1	社交网络图 .....	254
9.1.1	社交网络图告诉你和谁交朋友 .....	254
9.1.2	这几个基本概念你需要抓牢 .....	256
9.1.3	还有比本章任务更有趣的数据挖掘吗 .....	259
9.2	你还要装备几个评价指标 .....	260
9.2.1	社交网络大小 .....	260

9.2.2	社交网络关系的完备性 .....	261
9.2.3	节点实力评价 .....	262
9.3	全球某货物贸易中的亲密关系 .....	263
9.3.1	全球某货物贸易数据整合清洗 .....	263
9.3.2	分组和社交网络中心 .....	267
9.3.3	全球某货物交易圈：寻找各自的小伙伴 .....	270
9.4	中国电影演艺圈到底有没有“圈” .....	276
9.4.1	数据清洗与整形 .....	276
9.4.2	看看演艺圈长什么样 .....	279
9.4.3	谁才是演艺圈的“关系户” .....	281
9.4.4	用 Apriori 算法查查演艺圈合作的“朋友”关系 .....	283
9.4.5	给范冰冰推荐合作伙伴 .....	284
第 10 章	情感分析：一种准确率高达 90%的新方法？ .....	287
10.1	情感分析及其应用：这是老生常谈 .....	287
10.1.1	情感分析的用途 .....	287
10.1.2	情感分析的方法论 .....	288
10.1.3	有关情感分析的一些知识和方向 .....	289
10.2	文本分析的基本武器：R .....	290
10.2.1	RJava 包配置 .....	290
10.2.2	Rwordseg 包安装 .....	291
10.2.3	jieba 分词包安装 .....	291
10.3	基于词典的情感分析的效果好过瞎猜吗 .....	292
10.3.1	数据整理及词典构建 .....	292
10.3.2	分词整理 .....	297
10.3.3	情感指数计算 .....	299
10.3.4	方法评价：优、缺点分析 .....	300
10.4	监督式情感分析：挑选训练数据集是所有人心中痛 .....	301
10.4.1	TFIDF 指标 .....	301
10.4.2	构建语料库 .....	302
10.4.3	随机森林模型 .....	304
10.4.4	算法评估：随机森林应该建多少棵树 .....	308
10.5	一种准确率高达 90%的新方法 .....	316
10.5.1	拿来主义的启示 .....	316
10.5.2	情感词典和规则构建 .....	317

10.5.3 朴素贝叶斯情感分析器 .....	329
10.5.4 支持向量机 (SVM)、决策树等情感分析器 .....	330
10.5.5 如何选择支持 SVM 的核函数 .....	339
10.5.6 情感分类器方法评价 .....	343
10.6 谈谈情感分析的下一步思考 .....	344
<b>第 11 章 话题模型：很多牛人过不去的坎儿 .....</b>	<b>346</b>
11.1 话题模型与文案文本集 .....	346
11.1.1 任务仍然是以处理 dirty data 开始 .....	347
11.1.2 数据清洗 .....	348
11.2 话题模型中几个重要的数据处理步骤 .....	350
11.2.1 中文分词 .....	350
11.2.2 数据整型 .....	352
11.2.3 怎样设定“阈值” .....	353
11.3 上帝有多少个色子：话题数量估计 .....	356
11.3.1 通俗地说一遍话题模型 .....	356
11.3.2 主题数估计与交叉检验 .....	357
11.3.3 如何使用复杂度、对数似然值确定主题数 .....	362
11.4 LDA 话题模型竟然能输出这么多关系 .....	368
11.4.1 输出主题——词汇及其概率矩阵 .....	368
11.4.2 输出主题——文档归属及其概率矩阵 .....	369
11.5 话题之间也有社交（衍生）关系吗 .....	370
11.6 话题模型的几个强大衍生品 .....	372
11.6.1 话题模型提取特征词 .....	372
11.6.2 三种方法确定聚类的类数和文本层次聚类 .....	373
11.6.3 漂亮的文本聚类树和批量绘制大类词云图 .....	375
<b>第 12 章 排名就是简单的推荐系统吗？ .....</b>	<b>378</b>
12.1 全球宜居城市综合实力排行 .....	378
12.1.1 综合实力排行：专家法 VS 数据驱动法 .....	379
12.1.2 怎么比较两个排名结果 .....	382
12.2 协同过滤推荐系统 .....	383
12.2.1 基于商品的协同过滤系统 (ItemCF) .....	386
12.2.2 基于用户的系统过滤系统 (UserCF) .....	388
12.2.3 推荐系统效果评比 .....	390

第 13 章 生物信息学中的数据挖掘案例 .....	392
13.1 生物信息学与 R 语言 .....	392
13.2 生物信息学中常用的软件包 .....	392
13.2.1 软件包简介 .....	392
13.2.2 数据表示方式——对象类 ( class ) .....	393
13.2.3 生物信息学 R 包简介：Bioconductor 和 CRAN .....	393
13.2.4 ape 包 .....	394
13.2.5 读懂你的对象 .....	404
13.2.6 修改工具包中的函数以适应新情况 .....	407
第 14 章 产品化：关于内存、速度和自动化 .....	411
14.1 不同终端调用、自动化执行 R 脚本及参数传递 .....	411
14.2 与速度、内存、并行相关的程序优化 .....	414

---

轻松注册成为博文视点社区用户 ( [www.broadview.com.cn](http://www.broadview.com.cn) ), 扫码直达本书页面。

- 下载资源：本书如提供示例代码及资源文件，均可在 [下载资源](#) 处下载。
- 提交勘误：您对书中内容的修改意见可在 [提交勘误](#) 处提交，若被采纳，将获赠博文视点社区积分 ( 在您购买电子书时，积分可用来抵扣相应金额 )。
- 交流互动：在页面下方 [读者评论](#) 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口： <http://www.broadview.com.cn/33861>





# 第 1 章

## 万事不只开头难

### 1.1 工欲善其事，必先利其器：安装

R 语言是什么？是一门统计语言，这是认识 R 语言的基础，它生来就是做数据统计、数据分析、数据挖掘工作的。换句话说，SPSS 或者 Excel 能做的工作 R 都能做，但是 R 能做的工作用 Excel 完成简直是无法想象的，说这一点也是为了让大家在阅读本书和使用 R 时忘记那些用鼠标点点的傻瓜式操作，真正熟练之后你就会忘了那些傻瓜式的东西。最近随着大数据的兴起，R 语言用蔚然成风形容亦不为过，不信可以去各大招聘网站搜索数据挖掘或数据分析便知，或者在 Stackoverflow 搜索“Data Analysis”，统计一下相关问题的数量就可以了，一般使用的人数越多，针对这门语言的问题也就越多。

R 语言不是什么？R 不是通用语言，不要用 R 做开发的工作，此处开发工作指 App、网页、BI 网页报表等，虽然也有一些 R 语言包能够完成简单的网页报表开发，比如后面讲到的 shiny 包，但是这些东西基本是小儿科的花架子，上不了台面，前台系统真的还是需要 Java 或者 Python 开发。我建议，如有志于做数据挖掘工作，应该把自己的重心放到后台和统计学上。

不要想着一个工具就能包揽所有的工作，优秀团队的特征不是什么都能做，而是专业，即专业的分工。

R 是一款开源软件，它的功能包数量有 5000 多个，每个包囊括五花八门的任务，所以你能想到的大部分功能别人早就写好了，免费等着你用，用一句话来结束对 R 的介绍：R 让分析更便捷，单位代码产量高。何为单位代码产量高，即写同样多的代码，R 能完成更多的工作。

#### 1.1.1 安装 R 和 RStudio

RGui 是什么？简单地说就是 R 的编译器。选择 R 的理由是什么？开源、简单、代码优美，好吧，

## 探寻数据背后的逻辑：R 语言数据挖掘之道

这是我的理由。既然选定了 R，怎么获取 RGui，答案很简单，去官网。R 官网的链接为 <https://www.r-project.org>，打开之后点击“download R”，页面跳转后随意点一个链接，个人习惯是点击“China”下的第一个链接。

根据你使用的系统选择对应的 RGui，Linux、Windows 或者是苹果系统的。如果是第一次安装，则点击“install R for the first time”，然后按照提示下载对应的版本即可，下载后按照默认选项安装，至此 RGui 的安装已经完成。

安装了 RGui 以后就已经可以使用 R 了，但是为了更加方便地使用 R，推荐安装 Rstudio，它是目前 R 最漂亮实用的编辑器，编辑器更适合批量写代码，因为刚开始学的时候可能在 RGui 中一句一句地执行代码，但是如果你想将自己的代码保存下来，又不想一句一句将执行过的代码复制保存，很简单，用编辑器。Rstudio 是依赖于 R 的，所以安装 Rstudio 前一定要安装对应版本的 R 软件。

这些软件的使用基本上很简单，但是需要说明的是，在选择安装语言时尽量选择英文，原因很简单，你和世界沟通的语言是英文。

打开 RStudio 后，点击“File”新建 R 脚本，然后你会发现 RStudio 的界面可以分为四个部分，默认状态下左上角是编辑的脚本，左下角是 Console 控制台（相当于 R 的终端界面），跑的代码和打印的结果都会在左下角显示，右上角是环境和历史，执行的历史代码和数据对象都会在右上角显示，右下角是帮助和绘图显示区。

我们使用 `rnorm` 产生 1000 个随机值并绘制散点图，RStudio 的右下角显示所绘制的散点图，如图 1-1 所示。

```
plot(rnorm(1000))
```

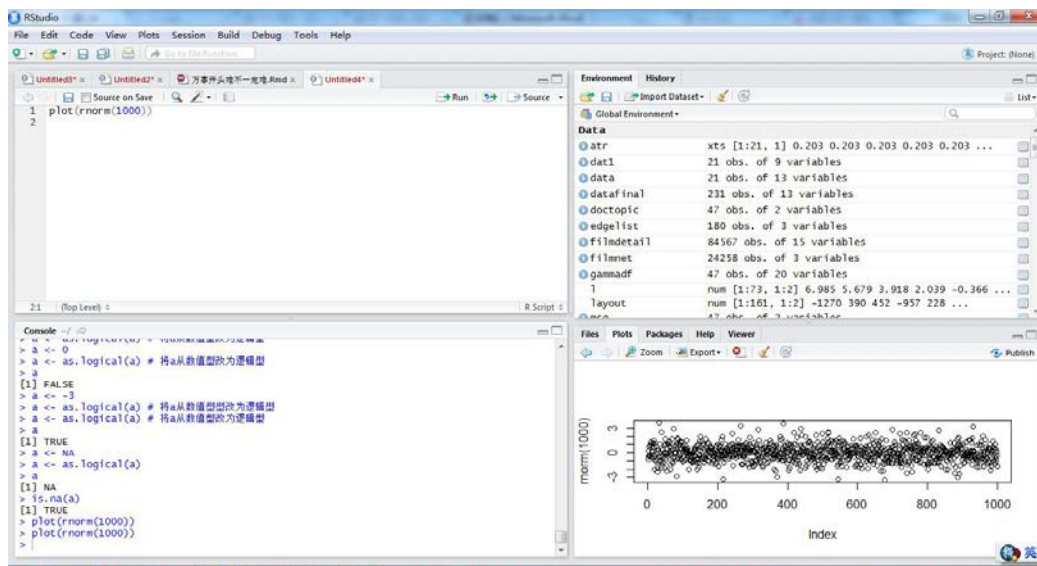


图 1-1

### 1.1.2 安装数据包

R 包可以认为是一些功能模块的集合，要 R 实现某些功能，比如特殊的模型、数据处理，就需要安装加载相应的包，这样可以帮助你节省自己开发某些功能点的时间和代码量，至于怎么找到特定的包，方法是用 Google 搜索，将自己的问题写成英文，搜索是最快的方式，用百度也可以，但是随着你所学加深，你的问题可能在中文社区找不到答案，所以，作为一个合格的程序员必须有能力使用 Google 和英文。

刚刚提到过，R 的数据包至少有 5000 多个，也就意味着不需要自己实现功能，直接引用 R 的数据包就可以实现很多复杂的功能。R 语言自带的包比较少，单纯地仅仅使用 R 的基础包不仅效率低下，而且函数也比较低效，初学者容易犯的一个错误就是没有在基础包里找到自己想要的函数，就马上动手写一个函数，殊不知动手搜一下相关的包，可能比自己写函数更加方便，更重要的是新手写的函数基本上不太实用。若使用基础包以外的包则需要先安装，放心，大部分是免费的。

#### 1. 在 RGui 下安装包

打开 RGui，点击“程序包”，选择“安装程序包”选项，设定 CRAN 镜像，源镜像里存储了很多经过审查且规范的包，个人习惯选取以 China 开头的镜像，之后选取要安装的扩展包即可（见图 1-2）。此处需注意一个问题，有时因为网络或者其他因素会导致数据包安装失败，这时候该怎么办？很简单，换一个镜像试试。

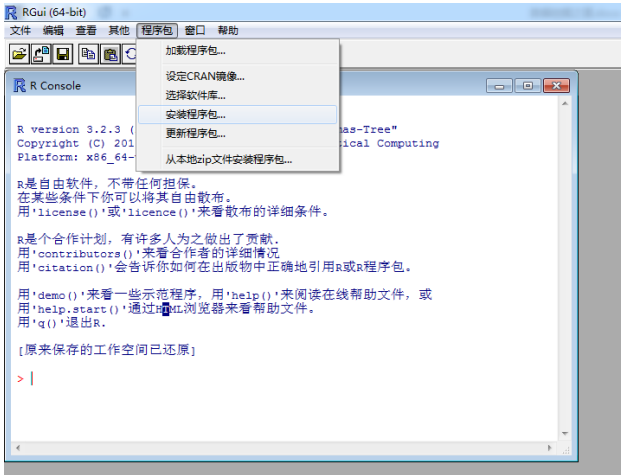


图 1-2

#### 2. 在 RStudio 下安装包

坦率地说，因为 RStudio 是依赖于 R 的，所以你在 RGui 上安装数据包以后，在 RStudio 上是可以直接使用的，但是通常情况下，如果学会了使用 RStudio 就会忘记 RGui，在 RStudio 中点击右下

侧的“Packages”，在“Packages”的左下方有个“install”按钮，点击该按钮即可安装包。默认是在线安装，在空白行中键入你要安装的包名，然后点击“install”按钮即可。

### 3. 通用方式安装

下面介绍一个取巧的方法，比如我现在更换了电脑，重新安装 R 后，且最好两台电脑上使用的 R 版本一致，一个一个下载对应的 R 数据包很麻烦，这时可以将原电脑 R 软件“library”文件夹下的文件复制到新电脑 R 软件“library”文件夹下，这样就不用重新安装数据包了。比如我的“library”文件夹路径是<C:\Program Files\R\R-3.2.3\library>（安装 R 时采用默认的路径安装），此时进入该路径复制文件夹下的所有文件，即可得到原电脑中的所有安装包。但是不建议这么做，因为会出现 Bug，特别是在 Linux 环境下，这里写出来是为了告诉大家不要图省事而使用这种雕虫小技。

无论是在 RGui 还是在 RStudio 中，都存在一种通用的方式，键入以下代码。

```
1 install.packages('你的包名')
```

将包名替换成你想要安装的包即可，举个例子，现在要从一批数据中寻找是否存在某种关联规则，拟定采用 Apriori 算法。Apriori 算法在 arules 包中，键入下列代码，即可安装 arules 包。

```
1 install.packages('arules')
```

install.packages 函数用来安装包，包名称用英文双引号或者单引号括起来。什么是函数？函数就是具有某种功能的小程序。如果是非匿名函数，那么每次调用程序时，只需要依据函数名称调取即可，比如 install.packages 就是一个安装包的小程序，它背后可能有一堆的代码，但是我们调用时只需要依据函数名称即可，英文小括号表示函数名称的结束，并在其中进行参数赋值，就如同  $f(x) = 33 + x$  一样，你要想获得输出结果，就要告诉函数  $x$  值为多少， $x$  就是参数， $f(x) = 33 + x$  就是函数，只不过这里将数学语言写成了计算机方程式。

### 4. 离线安装

有些包是不能直接使用上面的方式安装的，而且有些情况也无法完成在线安装，比如给一台未联网的计算机安装包就没办法直接安装，这就要提前下载相关的包，然后离线本地安装。方法如下：

#### （1）在 RGui 下离线安装

菜单栏中点击“程序包”，选择最后一项“从本地 zip 文件安装程序包”选项，之后定位到你要安装的数据包的路径即可。

#### （2）在 Rstudio 下离线安装

打开 Rstudio，点击“Packages→install”。选择“Package From File”，定位到你要安装的数据包的路径即可。或者使用如下代码在终端安装，指定安装包的存放路径即可。

```
1 install.packages("/root/ROracle_1.2-2.tar.gz")
```

其实，安装包还有很多讲究，比如在自动化程序下，程序员不知道 R 有没有安装某个必需的包，如果安装了就需要加载，如果没安装就要先安装后加载，需要一个自动判断的过程，这种情况将在第 14 章中进行讲解。

### 1.1.3 数据包加载、卸载、升级，查看帮助文档

在 R 里面使用非基础包是需要加载的，如果没有加载，程序就会提示“找不到所用函数”。加载数据包的方式有两种，library 和 require。library 和 require 究竟哪个好用？正所谓“萝卜青菜各有所爱”。

#### 1. 用 library 加载安装包

library 是我个人最常用的加载包的函数，这只是个人偏好而已。

用 library 加载包时，将包的名称用英文双引号或单引号括起来即可执行，当然直接书写包的名称也可以完成加载，执行后就将用于进行关联规则分析的包 arules 加载进内存了，代码如下。

- library 加载包

```
1 library("arules")
2 library(arules)
```

#### 2. 用 require 加载安装包

require 函数也可以加载数据包，代码同样简单。如下所示。

- require 加载包

```
1 require('arules')
2 require(arules)
```

但 library 和 require 还是有区别的，在后面的章节有可能用到它们的区别之处，特别是上面提到的自动化状态下，就要视情况选择使用二者之一。

- library 和 require 的区别

```
1 libraryreturn <- library('arules')
2 libraryreturn
3 requirereturn <- require('arules')
4 requirereturn
```

library 返回的是加载包的名称，包括依赖包，所谓依赖包就是在使用某些包时还需要加载其他相关包，而 require 返回的是一个逻辑值，即布尔数值，加载成功返回真 (TURE)，加载失败返回假 (FALSE)。

#### 3. 卸载数据包

你这是逗我吗？刚装上又要卸载？其实包的卸载真的没啥用处，包如果不加载到内存就是没用的东西，既不会影响速度也不会占内存，唯一的缺点就是占点硬盘空间。但是如果非要卸载，这里提供一种最简单的方法，即找到 R 的安装目录，在“library”文件夹中直接删除以相应数据包命名的文件夹即可。

#### 4. 升级数据包

数据包写好以后是一成不变的吗？数据包也会像软件一样更新，一个高效功能更加全面的版本

替代一个原始版本很正常，所以大家在使用某些包时要注意自己所选择的版本号，有些包更新之后相关函数的变化很大，甚至连名称都会改变，所以在更新数据包时一定要注意到这一点。

在 RStudio 下可以点击右下栏的“Packages”下的“Update”完成升级，在 RGui 中可以点击“程序包”选择更新程序包，或者直接重新安装就可以升级到最新版本。

也可以使用下面的函数对数据包进行升级。

```
1 update.packages()
```

### 5. 如何查看安装包说明文档

有时候安装了数据包之后需要查看一下包的说明文档，比如包里有哪些函数、函数的使用案例、包的作者等。

输入两个英文问号和包名称，即可返回 Vignettes 页面（Rstudio 在右下栏），点击相应的 PDF 文件就可以查看到包的说明文档，一般经过审查的官方源上的包，说明文档都是英文。

- 查看包的说明文档

```
1 ??arules
```

### 6. 查看函数的帮助文档和用例

R 通过查看数据包的说明文档，可以了解到数据包包含哪些有用的函数，而查看函数的帮助文档有更简单的方法，如下所示。

```
1 * help(函数名)
2 * ?函数名
```

在终端输入 help 函数，并将函数名作为参数，就可以查看该函数的帮助文档，或者在终端输入英文问号加函数名同样能看到函数的帮助文档，包括函数的描述、参数说明、一些计算细节还有相关文献，最后还有用例。有一点可以肯定，R 的函数说明文档要比 Python 更加详细，这也是我认为 R 的数据挖掘模块要比 Python 更加专业的原因，在高级语言中，如果没有审核机制，真的是一场灾难，你只能相信自己。例如查看 dim 函数的帮助文档，如下所示。

- 查看函数帮助文档

```
1 help(dim)
2 ?dim
```

看到函数的相关信息之后，想知道怎么样使用，可以阅读文档的后面的用例，当然你也可以在控制台查看函数如何使用。代码如下所示。

- 查看并执行函数用例

```
1 example(dim)
```

dim 函数用于查看一个矩阵或数据框的行列数，或者改变矩阵或数据框的行列数。在这里表达一下对 R 数据包开发者的敬意，先不说开发一个包有多难，仅准备所需的说明文档和用例就已经十分烦琐，能够有如此毅力的开发者值得我们尊敬。

### 1.1.4 什么样的 R 包值得相信

使用 R 语言也已经五六年了，但是我一直在回避思考一个问题，因为 R 和 Python 这类高级语言的很多应用模块和包是别人开发的，那么问题来了，怎么知道一个 R 包是否真正正确地解决了问题，问题的本质是我为什么相信某个包的开发者的能力，他会不会有隐藏的漏洞。通常我们使用的 R 包的来源无外乎三个：官方 CRAN、Bioconductor 和 Github。

与 Github 相比较，来自 CRAN 和 Bioconductor 的包更加可信，特别是来自 Bioconductor 的包。

首先，在 CRAN 和 Bioconductor 中包含了开发者讨厌而又必需的部分文件，例如说明文档、简介和一些标准的帮助代码，这让人相信他们的代码并不是为了临时应急而出的产品。其次，CRAN 和 Github 的区别还在于 R 的官方 CRAN 是用户导向的，必须保障用户的使用品质，而 Github 是开发者导向的平台，一切为了开发者更加方便，与 Python 的模块比较，这也是我更相信 R 包的原因。

一般我会在以下情况使用新的 R 包：

- (1) 为了迅速解决问题
- (2) 为了分享数据分析过程
- (3) 正在使用的包的依赖包

碰到以上三种情况，我可能要加载新包，如果我正在使用的包或将要使用的包废掉了，对于闹着玩的时候还没什么，但是一旦必须要重写、重测试时，那就头大了。因此对 R 包的信赖是视工作和项目情况而定的，一般我通过以下三条准则确定一个 R 包是否值得信赖。

- (1) 开发者优先
- (2) 来源鄙视链
- (3) 其他间接数据

每一条准则会在下面详细解释，之所以建立这三条准则有两个原因：首先保证我使用的包能正确地工作；其次如果包废了，至少可以保证版本更新时错误会被修复或者能够得到开发者的帮助。

- (1) 开发者优先

如果包的开发者开发过很多著名的包、开发过用户非常广泛的包或者开发过著名的软件，另外又能迅速对用户的问题做出响应，那么我会选择相信这个包。比如 Brian、Hadley、Jenny、Rafa，如果是他们开发的包，即使在一个非常不靠谱的链接上，我也会选择应用，因为他们开发的包大多会被选入 CRAN 或者 Bioconductor，会中断的概率非常小，即使没有最终发布，他们也会给出令人信服的原因。

- (2) 来源鄙视链

对于那些我一点都不了解的开发者，实在无法把他们的包排在优先位置。特别是对于功能相似的两个包，我只好采用来源鄙视链的选择方式强制选择，这里使用鄙视链有点过分，因为每一个开发者都值得尊重，工具没有高下之分，顺手就好，正如《卖油翁》所言“惟手熟尔”。首先我会优先选择 Bioconductor 中的 R 包，因为 Bioconductor 要求开发者提供比 CRAN 更加详细的说明文档

( vignette ), 其次是 CRAN, 它要求 R 包开发者进行代码自动检查 ( R CMD CHECK ), 并且有一套内容审核程序, 最后才是 Github 中的包。

虽然上述规则并不完美, 但是经过上面的筛选至少能说明开发者看待自己成果的态度, 如果一个开发者把包提交给了 CRAN/Biocnductor, 至少说明他们为此付出了更多的努力, 他们强迫自己进行了代码自动检验和说明文档写作, 这种内在的动力是值得信赖的。

### ( 3 ) 其他间接数据

除了使用优先级条件的条件外, 有时查看一下数据也能帮助你评估包及开发者。可以在 Bioconductor 查看该包的下载情况、测试情况等; 如果是来源于 CRAN 的包, 一般会去 Rstudio 查看下载状况, 因为在 R 官方网站中不显示用户对包的评价。如果一定要使用 Github 上的包, 那么首先要看一下它的关注 ( watch )、赞赏 ( star )、分支 ( fork ) 情况, 这些间接数据可以粗略估计一个包的情况; 还要看一看这个包的评论情况以及开发者对评论的反应, 如果开发者对包的状态和评论并不关心, 那么使用它的风险就比较大。还有一个小伎俩, 就是可以给开发者发送一条信息, 看看他的反应速度, 但是不建议使用这种方法骚扰开发者。

最后要观察一下开发者是不是在所有平台 ( Windows、Linux、Mac OS ) 都测试过包。这一点是有血的教训的, 我曾经使用过 googlevis 包, 结果做完项目才发现这个包必须在联网的情况下才能使用, 在离线状态下无法使用, 浪费了大量的时间和精力, 不知现在是否有所改善。如果开发者在各个平台都测试过而且使用者也给了很多赞, 开发者又能及时回答和处理问题, 那么这个包就更加值得信赖。

### ( 4 ) 总结

R 语言的生态系统要求使用者使用其他已有的包和函数, 而不是自己开发重写, 但是使用者也要保持轻松和安全的平衡, 尽量保持自己的依赖包名单足够短, 而不是毫无节制地扩张。

## 1.2 了解 R 的对象

所谓对象, 简单理解即命名之后的存储空间, 你可以对其进行查看、修改、增减等操作。其实 R 里面只有一个概念, 即对象, 所有的变量、数据、函数都是对象, 均需要加载到内存。但是这里将对象分为两个主要概念: 对象和函数, 便于读者理解。对象的概念仅限定在存储数据、模型、变量之内。

### 1.2.1 如何进行常见的算术运算

常见的算术运算无外乎加减乘除、对数、指数、开方等, 在 R 中使用 “<-” 符号进行对象赋值, 或者说是创建并赋值对象。如下所示。

- 常见算术运算



```

1 a <- 4 # 创建对象 a 并赋值为 4
2 a # 打印对象 a 的值
3 a <- 8 # 将对象 a 的值更改为 8
4 a
5 b = 7
6 d <- a - b # 对象 a 减去对象 b, 创建并赋值给 d 对象
7 print(d) # 打印 d 对象
8 print(A)
# Error in print(A) : object 'A' not found
9 a + b # a 加 b
10 a*b # a 乘以 b
11 a/b # a 除以 b
12 a^2 # a 的 2 次幂
13 a^0.5 # a 的 0.5 次幂, 即开平方
14 (a + b)^2/2 # a 加 b 之和的 2 次方然后除以 2
15 a <- log(x = a, base = 2) # 以 2 为底 a 的对数并重新将结果赋值给 a
16 log(a, 2)

```

在 R 里可以直接赋值, 也可以通过算式或函数进行赋值, 如“`d <- a - b`”和“`a <- log(x = a, base = 2)`”即为通过算式或函数赋值; 对象名称以大写或小写字符命名, 但 R 里对大小写是敏感的, 比如我们创建了对象 a, 打印对象 a 的值, 可以正常输出, 但是打印 A 却找不到对象, 所以若对象名称的大小写不一样, 则其所表示的意思也不一样; 从上面代码中看到, R 里面很多基础的运算函数都是使用符号表达的, 如加“+”减“-”乘“\*”除“/”幂“^”等, 但这类函数还是比较少的, 大部分是以英文命名的函数, 如对数函数 log, 函数一般包含至少一个参数 (也有特殊的), 比如 log 就有两个参数: base 用来指定对数的底, x 指定为谁求对数; 一般使用等号传参, 但是有时候开发者为省事就不写具体的传参过程, 而是直接书写, 这时 R 会根据参数与数值的先后顺序依次传参, 比如最后一行代码表示将 a 传递给参数 x, 将 2 传递给参数 base。

在 R 里创建并给对象赋值的符号为“<-”, 名称和值之间隔一个空格。也可以使用“=”创建对象, 但不建议使用这种方式, 二者大多数时候可以互换, 不过有些情况却有不同的含义, 其实“=”是一个传参符号, “<-”才是真正的创建和赋值符号, 如“`log(x = 8, base = 2)`”中的等号只是将 8 传给了 log 函数的参数 x, 并没有在内存中创建 x 对象, 打印 x 会报错; 但是“`log(x <- 8, base = 2)`”表示在内存中创建 x 对象并赋值 8, 然后将 x 传递给 log 函数的第一个参数 x, 这时打印 x 就会输出 x 的值为 8。

- 复制“<-”与传参“=”的区别

```

1 log(x = 8, base = 2)
2 x
#Error: object 'x' not found
3 log(x <- 8, base = 2)
4 x

```

R 里面“=”是传参, 那什么叫参数, 所谓参数就是函数正常工作时所需要设定的值, 比如这里

的 `x` 和 `base` 都是参数。参数一般分为输入的数据和设定的条件，例如 `x=8` 就是输入的数据，`base = 2` 就是设定的条件。输入的数据是指被操作的对象；设定的条件就是对数据进行操作符合哪些条件，比如是否移除缺失值、聚类聚几类；输入的数据和设定的条件统称为函数的参数，很多初学者可能将 “=” 误认为是比较是否相等的函数，其实不然，比较相等的函数为 “==”，如比较两个对象是否相等。

- “==” 与 “=” 的区别

```
1 a = 8
2 a == 7
3 rm(a)
4 a
```

上述代码中，第 1 行代码创建 `a` 对象并赋值为 8；第 2 行代码表示比较对象 `a` 和 7 是否相等，相等返回结果为真，不相等返回结果为假；第 3 行代码使用 `rm` 函数将对象 `a` 移除，再打印 `a` 时就找不到对象 `a` 了，这种方法在计算机内存吃紧时很有用，而且大家也要养成定期清除无用对象的习惯，以便节省内存空间。

就数据对象而言，R 里有四个比较常见的对象：向量、列表、数据框、矩阵，但在理解这些对象之前我们需要先了解三大数据类型：数值型、字符型、逻辑型。

### 1.2.2 R 语言的三大数据类型

最常见的数据类型为数值型，包括定距和定序两种，在 R 语言中创建数值型变量非常简单，直接将数值赋值给对象即可，如下所示。

- 数值型

```
1 a <- 3
2 b <- 3/2
3 mode(a)
# "numeric"
```

上面的对象 `a` 和 `b` 都是数值型对象，使用 `mode` 函数可以查看对象的数据类型，但是不要认为使用数字赋值给对象，对象的数据类型就一定是数值型，如下所示。

- 字符型

```
1 a <- "3"
2 is.character(a)
# TRUE
3 a/2
# Error in a/2 : non-numeric argument to binary operator
4 b <- "这是字符型"
5 mode(b)
6 nchar(b)
```

在 R 里使用英文双引号 " " 或者英文单引号 ' ' 标定字符数据，比如上述代码中，第 1 行代码将 3 用

英文双引号括起来，a 的数据类型就变成字符型，使用 `is.character` 函数判断对象 a 是否是字符型，如果是，则返回真，否则返回非；a 对象除以 2 会报错，因为字符数据没办法进行数学运算（见第 3 行代码）；将“这是字符型”赋值给对象 b，对象 b 就存储了这个字符串（见第 4 行代码）；函数 `nchar` 返回字符串对象的长度（见第 6 行代码）。

除了数值型和字符型外，还有逻辑型，逻辑型就包括两个状态：是或非，如下所示。

- 逻辑型

```
1 a <- TRUE
2 b <- FALSE
3 mode(a)
4 is.logical(b)
5 d <- T
```

“是”使用全部大写的 `TRUE` 表示，“非”用全部大写的 `FALSE` 表示，且不能用引号，否则就变成字符型了，当然写大量代码时为了节省敲击键盘的次数有时候是和非也可以缩写为首字母 `T` 和 `F`，但必须大写。其实理解数据类型很简单，但是在实战中经常用到的是数据类型之间的转化，例如你读入一个全是数值型的数据集，却不能进行算术运算，可以肯定你的数据里有些因素导致全部的数据变成字符型或者逻辑型了，这里就需要进行数据类型之间的转化，如下所示。

- 数据类型之间的转化

```
1 a <- "3"
2 b <- T
3 c <- 3
4 a <- as.numeric(a) # 将 a 从字符型转化为数值型
5 mode(a)
6 b <- as.numeric(b) # 将 b 从逻辑型转化为数值型
7 mode(b)
8 c <- as.character(c) # 将 c 从数值型转化为字符型
9 mode(c)
10 mode(a)
11 a <- as.logical(a) # 将 a 从数值型转化为逻辑型
12 a <- 0
13 a <- as.logical(a) # 将 a 从数值型转化为逻辑型
14 a
15 a <- -3
16 a <- as.logical(a) # 将 a 从数值型转化为逻辑型
17 a
18 a <- NA
19 a <- as.logical(a)
20 a
21 is.na(a)
```

上述代码中，使用 `as.character` 函数将其他数据类型的对象转化为字符型；`as.numeric` 函数将其他数据类型的对象转化为数值型；`as.logical` 函数将其他数据类型的对象转化为逻辑型，请注意当非

零数值型对象转化为逻辑型时都是 TRUE，只有 0 会被转化为 FALSE；另外 NA（不带引号，带引号的是字符）在 R 中表示缺失值，这里的空值指 not available，使用 is.na 函数判断一个对象是否是缺失值，其实 R 里的函数是很有规律的，比如做判断的函数常见的类型包括：is.xx、==、which、while、if 等，比如行使转化功能的函数经常以 as（作为）开头，形如 as.xx。

### 1.2.3 向量及其运算

R 语言中最常见的三种数据类型是数值型、字符型和逻辑型，但其实 R 语言里面没有基础的数据类型，所有仅包含一个数据点的对象都被看作长度为 1 的向量。向量由一系列类型相同的有序元素组合而成，从定义上理解向量必须包括两个属性：数据类型（mode）和长度。

- 向量

```
1 a <- c(T, F, T) # 创建逻辑型向量
2 b <- c(1, 2, 3) # 创建数值型向量
3 d <- c("a", "b") # 创建字符型向量
4 n <- c(d, b) # 强制统一类型
5 mode(n)
6 m <- c(a = 1, m = 2, b = 15)
7 names(m)
8 names(m) <- c("b", "h", "d")
```

上述代码创建了三不同数据类型的向量，c 函数就是 combine 的缩写，应该是 R 里面用到最多的基础函数之一，它的作用是将数值捆绑为向量或者 list，比如它将字符"a","b"捆绑成向量 d；也可以将两个向量 d 和 b 捆绑为向量 n，但是 d 为字符型向量，b 为数值型向量，捆绑后 n 的数据类型被定为字符型，所以 c 函数会对数据类型不同的值统一进行强制转化；向量中的元素除了具有自身的数值，还可以具有名字，比如向量 m 的元素，使用 names 函数可以查看修改（最后一行代码）向量元素的名称。

- 向量的筛选

```
1 x <- c(1, 34, NA, 54, 78)
2 length(x)
3 x[1:3]
4 x[c(1, 4, 5)]
5 x[c(T, F, T, F, T)]
6 x == 34
7 x[x == 34]
8 x[x > 50]
9 is.na(x)
10 !is.na(x)
11 x[!is.na(x)]
12 x[is.na(x)] <- 0
13 x
14 x[1] <- 2
```

```
15 x[x > 70] <- 70
16 ?'$'
```

有关变量的操作最常见的就是筛选和向量化计算，上述代码中，第1行代码创建了一个包含缺失值的向量 `x`；第2行代码使用 `length` 函数计算向量的长度，就是向量有几个元素，这里要注意，缺失值也会被计数；第3行代码筛选 `x` 的前三个元素，注意 R 里面筛选的函数有两种：中括号 `[]` 和美元符号（`s4` 对象可以用 `@` 筛选），查看它们的帮助文档在终端输入“`?$`”即可，前者对向量、list、数据框、矩阵都适用，只不过方式略有不同，但美元符号 `$` 不能应用于向量筛选，仅能用于 list、数据框等高级数据结构；第4行代码筛选编号为 1、4、5 的元素；第5行代码按照逻辑值提取元素，逻辑值的顺序和元素顺序一一对应，逻辑值为真时提取相应的元素，为假时不提取；第6行代码判断 `x` 的元素是否等于 34，其结果是一个和 `x` 等长的逻辑值序列；第7行代码使用“`x == 34`”产生的逻辑值提取 `x` 的元素，即提取值等于 34 的元素；第8行代码提取值大于 50 的元素；第9行代码判断 `x` 的值是不是缺失值，产生一个和 `x` 等长的逻辑序列；英文感叹号“`!`”在 R 里指非的意思，即将逻辑值反转，原来为真，使用“`!`”后变成了假，反之亦然；第10行代码使用“非”函数“`!`”将 `is.na` 判断的逻辑型结果逆转；第11行代码即筛选出 `x` 中不是缺失值元素的，从而剔除了缺失值；第12行代码筛选出 `x` 中所有的缺失值并将其赋值为 0，输出 `x` 就会发现 `x` 中的缺失值被替换成了 0；第14行代码将第一个元素替换为 2；第15行代码将所有大于 70 的元素替换为 70。

与向量相关的除了筛选，就是向量化运算，筛选是最常见的操作，毕竟一个 `select` 语句就成了一门数据库语言：SQL，可想而知，在数据处理过程中“筛选”这类操作多么常见。而向量化运算可能是 R 里面最重要的概念了，如果这个概念不能植入你的 R 语言编程思维中，那么你可能总是比别人慢一拍。

R 是一种高级语言，R 语言的程序需要解释为更低级的语言，如 C 和 C++ 后再执行，低级语言的速度要明显比 R 语言快。R 里的所谓向量编程，简单理解就是将多次调用函数的程序改成一次调用，比如常见的循环，有可能是多次调用同一个函数，但调用的程序需要被解释为低级语言，每次解释都要耗费时间，调用的次数越多，解释的次数就越多，那么耗费的时间也越多，而向量化运算将多次重复的输入作为一个整体输入，仅调用一次函数，也只解释一次，这样就节省了大量的解释时间，如图 1-3 所示。

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

$$\begin{aligned} 1 + 1 &= 2 \\ 2 + 2 &= 4 \\ 3 + 3 &= 6 \end{aligned}$$

图 1-3

图 1-3 上面是两个向量作为整体传递给 `sum`（求和）函数，也就只调用一次 `sum` 函数，后台仅解释一次；而下面是将两个向量拆成一一对应的具体数值，使用循环调用三次 `sum` 函数，解释器

就要解释三次。如果是两个上亿级别的向量相加，那么后者可能要半天时间才能完成。

- 向量化计算

```
1 a <- 1:3
2 a
3 a <- 1:100000 + 1
4 head(a, 10)
5 tail(a)
6 system.time(m <- sqrt(a))
# 用户 系统 流逝
#    0    0    0
7 tail(m)
```

在 R 里产生一个连续的公差为 1 的序列很容易，只需要在起始数之间添加一个英文的冒号 (:) 就可以了，如上述代码中，第 1 行代码产生一个 1 至 3 的序列；第 3 行代码产生一个 1 到 10 万的序列并给每一个值再加上 1；第 4 行代码查看 a 的前 10 个值，有些读者经常直接查看对象的所有元素，比如现在 a 里面包含 10 万个数据点，直接查看会非常麻烦，head 函数可以查看向量、数据框、矩阵的前几行数据，可以选择设置具体查看几行，这里设置为 10；tail 的英文是尾巴的意思，就是查看最后几行的数据，也可以设置查看具体的行数，一般默认 6 行；各位读者一定要养成检查数据的习惯，通过 head、tail 函数检查数据是最简单便捷的方式，二者相比我更喜欢用 tail，一般前几行数据不容易出错，后几行数据更具有代表性，因为越往后越有可能出现因特殊情况而导致的错误，所以我常常查看后几行数据；然后我们需要对 a 中的每个元素进行开方计算，如果采用向量化计算，可以直接将 a 作为一个参数丢给开方函数 sqrt，这样耗时多久呢？我们将这句代码放在 system.time 函数中，system.time 函数用于计算并返回执行代码所需的时间，向量化后，仅仅调用了一次 R 语言的 sqrt 函数，后台也仅解释了一次，所以速度非常快，时间几乎为零。

- 非向量化计算

```
1 m <- vector()
2 system.time(for (i in 1:100000) {
3   x <- sqrt(a[i])
4   m <- c(m, x)
5 })
6 tail(m)
```

如果采用非向量化的计算方式，即对 a 的每一个元素调用一次 sqrt 函数，就要写一个循环，对每一个元素开平方。将循环的代码块包括在 system.time 函数内就可以了，第 1 行代码先产生一个空向量 m，用于存放开方结果；R 里面的 for 循环将在 1.4 节详细讲解这里的意思，即如果 i 在 1 到 10 万这个序列内，就执行花括号内的代码。R 里 for 循环的结构：for、小括号、花括号。小括号用于指定循环的范围，花括号指定循环的具体操作，千万记住花括号里的内容每一次都会重复执行，花括号内第 1 行代码指定对 a 的第 i 个元素开方，然后将结果赋值给对象 x，第 2 行代码将 x 和 m 捆绑在一起再赋值给 m，这样随着 i 从 1 到 10 万重复循环，m 里就包含了 10 万个开方结果，此方法和向量化计算得到的结果一样，但是所用的时间差异真是让人大跌眼镜，这就是 R 语言向量化计算和非

向量化计算最大的区别之处。

如果读到这里仍然不理解向量化计算，就请记住这句话：尽量将整体作为一个处理单位，尽量规避一切可以规避的循环。这样即使你不理解向量化计算也知道怎么做，至于 R 代码的速度和向量编程的缺点我们将在第 14 章中详述。

## 1.2.4 因子变量鲜有人知的秘密

字符型变量的本质任务是为了分类，比如是  $a$  不是  $b$ ，其实质是一种定性的变量，而数值是定序定距的变量，也就是说数值型一般属于定量类变量，有读者会提出连续型变量也可以做分类，但是不要忘了连续型变量做分类通常需要设置区间，比如 1~3 这个区间内为类别 1，3~4 这个区间内为类别 2，这样做的实质是将连续型变量转化为字符型变量。有趣的是，我们经常碰到这样的情况，特别是在定性研究中，比如我们调研客户对某种口味（不要问我什么口味）的喜爱程度时，通常问卷设置如下选项：非常讨厌、讨厌、不讨厌、比较喜欢、喜欢、大赞，这些选项是字符型分类变量，但是它们又有定序的功能，这就用到一种新的变量类型：因子，即有序的分类变量；R 里面一个因子包括两个属性：因子值和因子水平。请看如下代码。

- 因子

```
1 a <- factor(x = c("非常讨厌", "不讨厌", "大赞", "讨厌", "比较喜欢", "大赞", "喜欢", "喜欢"), levels = c("非常讨厌", "讨厌", "不讨厌", "比较喜欢", "喜欢", "大赞"), ordered = T)
2 a
# [1] 非常讨厌 不讨厌 大赞 讨厌 比较喜欢 大赞
# [7] 喜欢 喜欢
# 6 Levels: 非常讨厌 < 讨厌 < 不讨厌 < 比较喜欢 < ... < 大赞
3 length(a)
4 levels(a)
5 levels(a) <- c("讨厌", "非常讨厌", "不讨厌", "比较喜欢", "喜欢", "大赞")
6 a
```

factor 函数用于创建因子对象或者将别的变量转换为因子，这里使用了其 3 个参数，比如我们调研了 8 个人对某种口味的喜好程度，就将这 8 个人的选择结果依次使用 c 函数捆绑为向量传递给参数 x，第 2 个参数 levels 用于设定因子水平，我们将问卷中的选项设置为因子水平，问卷中共 6 个选项，我们设置了 6 个因子水平，而且仔细观察会发现这些因子水平之间好像还存在一定的顺序，比如讨厌要好于非常讨厌，所以我们使用 ordered 参数告诉函数这个因子水平是定序的，它们之间存在程度的差异；输出 a 会发现 a 比常见变量多输出了一行 levels；length 函数和 levels 函数可以查看因子变量的长度和因子的水平，另外，levels 函数还具有修改因子水平的功能，比如你的程度感觉系统和别人不一样，认为非常讨厌比讨厌好点，可以使用 levels 函数重新修改对象的因子水平（倒数第 2 行代码）。

- 因子与字符向量的相互转化

```
1 a <- c("非常讨厌", "不讨厌", "大赞", "讨厌", "比较喜欢", "大赞", "喜欢", "喜欢")
```

```
2 mode(a)
3 a <- factor(a)
4 levels(a)
5 levels(a) <- c("非常讨厌", "讨厌", "不讨厌", "比较喜欢", "喜欢", "大赞")
6 a <- as.character(a)
7 is.factor(a)
```

对象之间的转化是最常见的工作，上述代码中，第 1 行代码产生一个字符型变量 `a`；第 2 行代码使用 `mode` 函数查看 `a` 的类型，为字符型；第 3 行代码使用 `factor` 函数（也可以用 `as.factor` 函数）将 `a` 转化因子；第 4 行代码查看因子水平，发现因子水平和实际上不一致，默认情况下，因子水平和字符在向量中的出场顺序一致；第 5 行代码使用 `levels` 函数将因子水平修改为正常的顺序；第 6 行代码使用 `as.character` 函数将 `a` 重新转化为字符向量；第 7 行判断对象 `a` 是不是因子，返回逻辑值。

在 R 语言中，有很多函数默认情况下会将字符变量转化为因子，而因子失去其利用场景之后会存在很多不必要的麻烦，我之前一直认为这种默认是一种非常愚蠢的做法，直到发现因子比字符所占的内存要小很多（和字符长度和因子水平的多寡有关），才觉得这种默认的高明之处。请看如下代码。

- 因子比字符变量更节省内存

```
1 a <- c("非常讨厌", "不讨厌", "大赞", "讨厌", "比较喜欢", "大赞", "喜欢", "喜欢")
2 a <- rep(x = a, times = 10000)
3 b <- factor(x = c("非常讨厌", "不讨厌", "大赞", "讨厌", "比较喜欢", "大赞", "喜欢", "喜欢"))
4 b <- rep(b, 10000)
5 length(a) == length(b)
6 object.size(a)
7 object.size(b)
```

上述代码中，第 1 行创建一个字符型向量；第 2 行代码使用 `rep` 函数将向量 `a` 复制 10000 次；第 3 行代码创建一个包含相同数据的因子变量 `b`；第 4 行代码将 `b` 复制 10000 次重新赋值 `b`；第 5 行代码检查 `a` 和 `b` 的长度是否相等，可以肯定 `a` 和 `b` 存储的数据相同，只不过变量类型不同；第 6 行代码查看变量 `a` 所占的内存，大约占用了 64 万比特，而因子 `b` 仅占用了 32 万比特，可见字符型变量在某些情况下比因子占用更多的内存，究其原因是因为其存储方式不同，因子变量的值是以 `integer`（整数）存储的，仅仅是因子水平使用了字符存储，如下所示。

```
1 b <- factor(x = c("非常讨厌", "不讨厌", "大赞", "讨厌", "比较喜欢", "大赞", "喜欢", "喜欢"), levels = c("非常讨厌", "讨厌", "不讨厌", "比较喜欢", "喜欢", "大赞"), ordered = T)
2 unclass(b)
```

因子变量和字符一样，不能用来计算而只能用来分类、计数，当有大量重复的字符变量时，可以将字符变量暂时存储为因子，以此减少内存压力。

上面用到了 `rep` 这个函数，下面讲述一下它的用途，`rep` 就是 `replicate` 的缩写，它的作用是将一个对象复制多次，但是复制有两种方式，其结果适用不同的应用场景，先看看是哪两种：

- `rep` 的两种形式



```
1 a <- c(1:3)
2 rep(a, times = 3)
3 rep(a, each = 3)
4 rep(a, 3)
```

上述代码中，第1行代码产生一个1至3的数值向量；第2行代码使用 rep 函数将向量复制3次，注意其复制结果是整体复制一遍三次；第3行代码将向量 a 的元素每个复制3次，和上一行结果的排列顺序是不一样的；第4行代码默认情况下 rep 函数是按照 times 复制的。按照 each 复制比较适用于后面用到的多因子方差分析或者为 mapply 创造索引，后面还会讲到。

## 1.2.5 矩阵相关运算及神奇的特征值

矩阵就是将一维的向量概念拓展到了二维，所以要求所有数据的类型必须一致，例如不能同时存在字符型和数值型。

- 矩阵

```
1 mymatr <- matrix(data = c(1:100), nrow = 25, ncol = 4, byrow = TRUE)
2 dim(mymatr)
3 a <- 1:4
4 b <- 1:4
5 m <- cbind(a,b)
6 n <- rbind(a,b)
7 g <- rbind(n, n, n)
```

上述代码中，第1行代码中的 matrix 函数用于创建矩阵对象，或者将其他类型的对象转化为矩阵，第1个参数 data 用于指定数据，nrow 和 ncol 用于指定矩阵的行数和列数，我们创建了一个 25 × 4 的矩阵，byrow 是一个逻辑值，设定数据是按行排列还是按列排列的，将其设置为 F，可以查看它们的差异；第2行代码使用 dim 函数查看矩阵的行列数；第3行和第4行代码创造 a、b 两个向量；第5行代码使用 cbind 函数将向量 a、b 按列捆绑为矩阵 m；第6行代码使用 rbind 函数将向量 a、b 按行捆绑为矩阵 n；cbind 和 rbind 这两个函数非常有用，因为在数据整理时经常将向量和数据框捆绑为新的数据，它们不仅能把向量捆绑为矩阵，而且能把矩阵捆绑为新矩阵（最后一行代码将三个矩阵捆绑为新的矩阵），或者把数据框捆绑为新的数据框。

- 矩阵的常见运算

```
1 mymatr <- matrix(data = c(1:20), nrow = 4, ncol = 5, byrow = TRUE)
2 trans <- t(mymatr) # 转置
3 mymatr - mymatr # 矩阵加减
4 trans*2 # 数与矩阵相乘
5 mymatr %*% trans #
6 diag(mymatr) # 取对角元素
7 rowSums(mymatr) # 对行求和
8 colSums(mymatr) # 对列求和
9 mymatr <- matrix(data = c(1:25), nrow = 5, ncol = 5, byrow = TRUE)
```

```
10 mymatr.eigen <- eigen(mymatr, symmetric = T)
11 mymatr.eigen
12 mymatr.eigen$values #特征向量
13 mymatr.eigen$vectors #特征值
```

上述代码中，第 1 行代码生成一个  $4 \times 5$  的矩阵；第 2 行代码将 mymatr 矩阵转置，所谓转置即将行变为列，将列变为行，原来的  $4 \times 5$  就变成了  $5 \times 4$  矩阵，函数 t 只有一个参数，用来指定需要转置的矩阵或者数据框对象，t 就是 transpose 的缩写；第 3 行代码求两个矩阵的差，矩阵之间的加减需为发生在同行同列的矩阵；求矩阵的乘积的函数为 %\*%，需要 a 为  $m \times n$  矩阵，b 为  $n \times k$  矩阵才能求积；diag 函数用于获取矩阵的对角元素；rowSums 和 colSums 分别用于按行或者按列求和；eigen 用于求矩阵的特征值和特征向量：mymatr.eigen\$values 提取矩阵的特征值，而 mymatr.eigen\$vectors 用于提取矩阵的特征向量，换句话说，mymatr.eigen 是一个表单，包括了矩阵的特征值和特征向量，1.2.3 节我们介绍筛选时讲了两个筛选符号：中括号 ([]) 和美元符号 (\$)，\$ 函数用于按名称提取矩阵、表单、数据框元素，这里用来提取表单 mymatr.eigen 中矩阵的特征值和特征向量，如果不理解矩阵和特征向量、特征值之间的关系，那么可以翻开数学课本了解一下，我建议常备一本高数书、一本统计学图书在身边比较好。

特征值和特征向量有什么用呢？最简单的一个例子就是数据降维，比如 PCA (Principal Component Analysis, 主成分分析)，什么是数据降维，简单说即是本来有 2000 列数据，相当于 2000 个分析维度，但为了简化过程，在尽量保持数据原有信息的前提下，降到 20 个分析维度。特征向量反映了线性变换的方向，特征值反映了各个方向变化的多少，而数据降维就是保留那些对数据内的方差影响最大的维度，即方向。换句话简单描述 PCA 中的主成分就是特征值较大的特征向量，后面我们将详细讲解。

到这里是不是觉得有点难了，不要觉得难了就拒绝前进了，现在不懂的东西后面还会详细讲解。其实 R 语言不难，它只是一门统计语言，但是结合统计学和数据就比较难了，可是生活不就是这样吗，万事开头难，然后中间难，最后结尾难，所以每当我觉得今天很难受时，想到明天更难受，顿时心里就会觉得轻松多了。其实很多数据学计算都已经在模型里进行了，现实中用到的矩阵知识并不多，但是了解模型的数学原理还是非常有必要的。

### 1.2.6 数据框及其筛选、替换、添加、排序、去重

1.2.5 节我们讲到矩阵是向量的二维扩展，因子矩阵的数据类型必须保持一致，但大多数时候的表只是一个二维表，里面的数据类型五花八门，所以这就需要引入一个存储这类数据的对象类型，就是数据框 (data.frame)，它对存储的数据类型没有要求，仅仅要求每一类具有相同的长度。

- 数据框与矩阵

```
1 a <- 1:3
2 b <- c("a", "b", "c")
3 m <- cbind(a,b)
4 m
```

```

5 d <- data.frame(a, b)
6 str(d)
# 'data.frame': 3 obs. of 2 variables:
# $ a: int 1 2 3
# $ b: Factor w/ 3 levels "a","b","c": 1 2 3
7 d <- data.frame(a, b, stringsAsFactors = F)
8 str(d)

```

上述代码中，第1行和第2行代码创造了两个向量，数据类型分别为数值和字符；第3行代码将向量 `cbind` 设为矩阵，输出矩阵你会发现 `a` 列的数据已经被加上了双引号，变成了字符；第5行代码的 `data.frame` 函数将多个向量捆绑为数据框；第6行代码的 `str` 函数用于查看 R 对象的结构 (structure)，可以发现 `d` 是一个数据框，包含了3行2列数据，两列数据分别是整数和因子，有读者会问：为什么 `b` 的数据类型由字符变成了因子？首先应该承认数据框中是可以存储数据类型的，然后数据框在默认状态下会改变数据的类型，即在默认状态下将字符型改变为因子型，原因之前已经讲过，在多数情况下因子比字符更加节省存储空间；倒数第2行代码，我们将函数 `data.frame` 的参数 `stringsAsFactors` 设置为假，即可拒绝将字符转化为因子，该参数用于设定是否将数据中的字符型的变量转化为因子，比较常用；通过 `str` 函数查看数据结构就会发现数据类型和原来的向量保持一致，但数据框要求各个变量的长度一样，你不能将长度为2的变量和长度为3的变量捆绑为一个数据框，R 将会抛出 “Error in data.frame(a, b): 参数值意味着不同的行数: 2, 3” 类似的错误。

- 常见的操作：筛选

```

1 myframe <- data.frame(a = c(1:3, 2:5, 8), b = c("a", "b", "c", "b", "c",
" d", "e", "f"), m = c(NA, "b", "c", "b", "c", NA, "e", "8"), n = 1:8,
stringsAsFactors = F)
2 str(myframe)
3 myframe[1:3, ] # 筛选前三行数据
4 myframe[, 1:2] # 筛选前两列数据
5 myframe[1:3, c(1, 4)] # 筛选第1列和第4列的前三行数据
6 myframe[, c("a", "m")]
7 library(dplyr)
8 select(myframe, b)
9 select(myframe, -b)
10 myframe[myframe$a == 1, ]
11 myframe[myframe$a > 3, c("b", "a")]
12 temp <- c("c", "a")
13 myframe[myframe$b %in% temp, ]

```

筛选是最基础的操作，无论是替换还是排序等都需要先将数据筛选出来，但越是基本的操作越需要专业的态度去把它们做好。上述代码中，第1行代码创建一个8行4列的数据框；第2行代码查看数据框的数据结构，发现其中包含两个数值型变量和两个字符型变量；第3行代码筛选前三行数据，前面我们讲过使用中括号筛选向量，其实矩阵和数据框的筛选和向量一样，只不过在中括号中间加了一个英文逗号“，”，用于标识行、列的筛选，筛选文本框和筛选矩阵是一样的方法；第4

行代码筛选前两列；第 5 行代码筛选第 1 列和第 4 列的前三行数据；以上都是按照索引编号筛选的，时间长了或者数据结构变量之后你可能不记得第 1 列代表的是什么数据了，这样的筛选代码的可读性不高，所以第 6 行代码使用行名称筛选了数据框的 a 列和 m 列，这样我们就不用记住 a 列是第几列，只要数据框中存在 a 列即可，建议大家以后写代码还是要按照这种方式筛选；那问题来了，如果我有 300 列，但是仅想剔除其中的一两列怎么办？dplyr 包中有一个 select 函数可以对数据进行按列筛选，第 8 行代码使用 select 筛选数据框的 b 列，select 函数的第 1 个参数指定数据集，后面的参数指定筛选的列名称，不加双引号；第 9 行代码在列名称前添加一个减号表示筛选除 b 列以外的其他列；第 10 行代码筛选出 a 列等于 1 的所有数据；第 11 行代码筛选出 a 列大于 3 所对应的 b 列和 a 列数据，myframe\$a > 3 等于做出了一个判断，生成了一个和数据框等长的逻辑值向量，大于 3 为真，小于等于 3 为假，再按照这个逻辑序列筛选行，且仅筛选出 b 列和 a 列数据；最后一行代码使用 %in% 进行匹配筛选，用 myframe\$b 中的元素去 temp 里面查找，返回一个和 myframe\$b 等长的逻辑序列，查到后返回真，查不到返回假，然后按照逻辑序列筛选。

- 常见的操作：替换修改

```
1 myframe <- data.frame(a = c(1:3, 2:5, 8), b = c("a", "b", "c", "b", "c",  
"d", "e", "f"), m = c(NA, "b", "c", "b", "c", NA, "e", "8"), n = 1:8,  
stringsAsFactors = F)  
2 myframe[is.na(myframe$m), "m"] <- 0  
3 myframe[myframe$b == "c", "a"] <- 1  
4 myframe[myframe$n <= 4, "a"] <- 1  
5 myframe[myframe$b == "c", "a"] <- myframe[myframe$b == "c", "n"]  
6 myframe2 <- data.frame(a = c(2:5, 2:5, 7), b = c("a", "b", "c", "b", "c",  
"d", "e", "f"), stringsAsFactors = F)  
7 myframe[myframe$b == "c", "a"] <- myframe2[myframe$b == "c", "a"]
```

筛选操作经常是为了可以将其替换或被替换，上述代码中，第 1 行代码产生数据框 myframe；第 2 行代码使用了按名称提取的筛选符号 \$，筛选出为缺失值的 m 列数据并将缺失值替换为 0，你可以在控制台输入 myframe 查看替换后的效果；第 3 行代码筛选出 b 列等于字符 c 所对应的 a 列数据并将之替换为 1；第 4 行代码筛选 n 列小于等于 4 所对应的 a 列数据并将之替换为 1；第 5 行代码筛选 b 列等于字符 c 所对应的 a 列数据并将之替换为 b 列等于字符 c 所对应的 n 列数据，有点拗口，将 a 列的数据替换为 n 列只不过加了一个筛选条件；第 6 行代码生成一个相似的数据框 myframe2；第 7 行代码 myframe\$b == "c" 作为条件产生一个逻辑序列，然后用这个序列筛选出所对应的数据框 myframe 中的 a 列数据，同时用这个序列筛选出 myframe2 中对应的 a 列数据，将前者替换为后者，需要注意的是 myframe 和 myframe2 必须是等长的数据框，也就是行数要相同。

- 常见的操作：添加

```
1 myframe <- data.frame(a = c(1:3, 2:5, 8), b = c("a", "b", "c", "b", "c",  
"d", "e", "f"), m = c(NA, "b", "c", "b", "c", NA, "e", "8"), n = 1:8,  
stringsAsFactors = F)  
2 temp <- 1:8  
3 myframe$new1 <- temp
```

```

4 myframe$new2 <- myframe$a/myframe$n
5 myframe <- transform(myframe, new3 = a/n)
6 library(dplyr)
7 myframe <- select(myframe, -contains("new"))
8 myframe[9, ] <- 1:4
9 myframe <- rbind(myframe, myframe)

```

上述代码中，第 1 行代码生成一个数据框；第 2 行代码生成一个临时向量；第 3 行代码使用美元符号在数据框中创建新列 new1，并将 temp 赋值给新列，这样就在 myframe 中添加了一个新数据列；第 4 行代码使用美元符号创建新列 new2，并将 myframe\$a 除以 myframe\$n 获得的结果赋值给新列 new2，通过终端输入 myframe 可以查看其变化；第 5 行代码的 transform 函数用于修改数据框，第 1 个参数指定数据，第 2 个参数是一个表达式，意思是在 myframe 中创建新列 new3，并将 a 列除以 n 列的结果赋值给新列 new3；第 7 行代码使用 select 函数将数据框中刚刚创建的三个新列删除，因为它们都包含 new 字符，所以使用 contains 参数获得包含 new 字符的列名称，然后使用“-”号将其剔除，这样数据框又恢复到原来的样子，想了解更多 select 的应用可以在终端输入“?select”查看用例；第 8 行代码在数据框 myframe 中添加第 9 行，并将第 9 行赋值为 1 至 4；第 9 行代码将两个数据框按行捆绑为一个新的数据框，等于给数据框添加了行数据，捆绑时要求数据框的列数和列名称一致。

- 常见的操作：排序相关的几个函数（1）

```

1 x <- c(77, 83, 95, 64, 32, 100, 94, 62)
2 sort(x, decreasing = T)
3 order(x, decreasing = T)
4 x[order(x, decreasing = T)]
5 rank(x)
6 cbind(x, rank(x))

```

上述代码中，第 1 行代码创建一个数值型的向量 x；第 2 行代码的 sort 函数对数据进行排序，它包括两个参数：第 1 个用于指定数据，第 2 个用于指定升降序，降序用 TRUE，升序用 FALSE，需要注意的是，sort 函数直接返回排好序的原数据值，这样其作用就大大弱化了，换句话说它仅能进行向量或因子的排序；而 order 函数能够对多列排序，参数可以有很多，基本上前面的参数都是用来指定数据的，然后 decreasing 指定升降序，order 函数返回的是原数据的索引号，必须排第一的 100 在原数据的位置索引号为 6，order(x, decreasing = T)的结果第一个就是 6；第 4 行代码查看原始数据排序后的结果，只需要将 order 函数的结果作为索引筛选一次即可；第 5 行代码为函数 rank 返回排序之后的秩；第 6 行代码将 rank 获得的秩和原数据捆绑在一起就知道 rank 得到的秩什么意思了，对，它就是名次编号。

- 常见的操作：排序相关的几个函数（2）

```

1 myframe <- data.frame(a = c(1:3, 2:5, 8), b = c("a", "b", "c", "b", "c",
"d", "e", "f"), m = c(NA, "b", "c", "b", "c", NA, "e", "8"), n = 1:8,
stringsAsFactors = F)
2 myframe[order(myframe$n, decreasing = T), ]

```

```
3 myframe[order(myframe$a, myframe$b), ]
```

知道了向量怎么排序才能理解数据框怎么排序，矩阵和数据框排序的方式是一样的。上述代码中，第 1 行代码创建一个数据框；第 2 行代码使用 `order` 函数对 `n` 列进行降序排列，`order(myframe$n, decreasing = T)` 获得了排序后每一行数据的索引编号，然后按照这个索引编号提取数据，即对数据框 `myframe` 按照 `n` 列降序排列；第 3 行代码稍微增加了点难度，即对数据框 `myframe` 按照 `a` 列和 `b` 列升序排列。

排序很重要，测试一个系统速度怎么样，大家比较公认的方法就是测试排序，当我们面对大数据时这些基础的排序函数速度可能会下降很多，我们在后面的章节中会讲述一些更加高效的方法。

- 常见的操作：去重

```
1 myframe <- data.frame(a = c(1:3, 2:5, 8), b = c("a", "b", "c", "b", "c",  
"d", "e", "f"), m = c(NA, "b", "c", "b", "c", NA, "e", "8"), stringsAsFactors  
= F)  
2 unique(myframe)  
3 duplicated(myframe$a)  
4 myframe[!duplicated(myframe$a), ]  
5 myframe[duplicated(myframe[, c("a", "b")]) , ]  
6 myframe[!duplicated(myframe[, c("a", "b")]) , ]
```

去重是另外一种非常基础的操作。上述代码中，第 1 行代码生成数据框；第 2 行代码使用 `unique` 函数对整个数据框去重，即保留那些不重复的行，`unique` 函数也可以用于向量的去重，它返回的是原数据去除重复的数据；第 3 行代码的 `duplicated` 函数判断 `myframe$a` 是否是重复数据，如果是则返回真，不是则返回假，即第一次出现返回假，再次出现返回真；然后我们在 `duplicated(myframe$a)` 的前面添加 “!”，函数就将原来的逻辑序列反转了，即第一次出现变为真，再次出现变为假，使用这个逻辑序列就可以筛选出不重复的数据（第 4 行代码）；同样第 5 行代码将数据框 `a` 列和 `b` 列同时作为判断依据，筛选出重复的数据；第 6 行代码恰恰相反，是要筛选出不重复的数据。

有关数据框的基础操作基本上就讲解完了，至于数据框的合并和关联，我们将在后面的章节单独讲解。

### 1.2.7 与数组（array）相比，表单（list）的用处更加广泛

数组（array）是向量或者说矩阵在三维上的延伸，矩阵是行列式，如果给矩阵添加一个轴，矩阵就变成了数据立方，而数据立方就是数组的外现形式，下面的代码形成一个数组：

- 数组

```
1 arr <- array(1:27, dim = c(3,3,3))  
2 arr  
3 arr[3,3,3] <- "a"  
4 arr
```

上述代码中，第 1 行代码创建了一个  $3 \times 3 \times 3$  的数组；第 2 行代码输出数组会发现就是存储了三个矩阵；第 3 行代码将数组的最后一个改为字符 `a`；输出后发现所有的数据都加了双引号，也就是

说数组强制将所有的数据转化为字符,这也就是数组在日常数据应用较少的原因,它远没有表单(list)用处丰富。

其实 list 的用处已经超越了上面讲过的其他对象,因为它被定义为对象的装载器,换句话说,即什么对象都可以丢到 list 里面存储下来。

- 表单

```
1 a <- 1:3
2 b <- c("a", "b", "b")
4 m <- data.frame(cbind(a, b))
5 names(m) <- c("x", "y")
6 alist <- list(a, b, m)
7 alist
8 names(alist) <- c("g", "h", "k")
9 alist$k # 使用美元符号筛选
10 alist[3] # 使用中括号筛选
11 class(alist[3])
12 alist[[3]]
13 class(alist[[3]])
```

上述代码中,第 1 行代码创建一个数值型向量;第 2 行代码创建一个字符型向量;第 3 行代码将向量 a、b 按列捆绑在一起,并转化为数据框;第 4 行代码使用 names 函数提取文本框的列名称,并赋值新的列名向量,这样数据框 m 的列名称就被修改成了 x 和 y;第 5 行代码将向量 a、b 和数据框统统捆绑为一个 list;我们看到表单 list 不仅包含了不同的对象(向量、数据框),同时也包含了不同的数据类型(数值、字符、因子);第 7 行代码修改 list 中元素的名称;第 8 行代码使用名称筛选 list 的元素;第 9 行代码使用我们熟悉的中括号筛选 list 元素,方式和 vector 一样,但是筛出来的是一个仅包含一个元素的 list,其实 alist 的第 3 个元素是一个数据框,想要筛出来数据框需要使用两层中括号,如第 11 行代码;第 12 行代码使用 class 函数查看双中括号筛出来的对象的类型,其结果是一个数据框。

## 1.2.8 如何进行数据结构之间的转化

前面用了不少的篇幅介绍 R 中的数据对象,我觉得无论对这些基础知识报以多么大的关注都不为过,因为这些是基本功,我刚开始工作时,自以为很了解 R,可是一旦做数据清洗工作,就捉襟见肘、寸步难行了,后来发现还是这些基础的知识不够扎实。以上这些对象是 R 中最为常见的数据结构,其他更加高级的数据类型和类(class)都是由这些基本数据作为函数的输入转化而来的。

其实使用一个函数最重要的是了解它的输入,它要求输入是数据框还是 list,数据框的结构是什么,弄明白这些再稍微了解下其他参数的意义,这个函数对你来说基本上不会有太大障碍,但首先要熟练常见的数据对象之间的转化。

今天我们就以 R 自带的鸢尾花数据集进行一次彻底的基础对象之间的转化,鸢尾花数据集可以说是一个经典的数据集,很多算法在演示时都是使用它。它一共包括 5 列 150 条观测值,共记录了 3

种鸢尾花的 4 个观测属性，经常被用来做分类算法的样例数据。

- 对象转化：数据框—list—矩阵—数据框

```
1 tail(iris)
2 str(iris)
3 iris$Species <- as.character(iris$Species)
4 str(iris)
5 temp <- unclass(iris) # 数据框转化为 list
6 str(temp)
7 temp <- do.call(cbind, temp) # list 转化为矩阵
8 temp <- data.frame(temp, stringsAsFactors = F) # 矩阵转化为数据框
9 str(temp)
10 temp$Sepal.Length <- as.numeric(temp$Sepal.Length)
11 temp$Species <- as.factor(temp$Species)
```

上述代码中，使用 `tail` 函数默认查看数据末尾的 6 条数据；`str` 函数用于查看数据结构，可以看到 `iris` 数据集的类（class）为数据框包含 150 条数据，5 个变量，其中前四个变量是数值型（num），最后一个变量为因子。这里要说一下什么是类，类就是对象的官方说法，类一般包括属性（attr）和方法（method），当然有时候也不一定有，比如数据框是一个类，而 `iris` 数据集的数据框形式就是这个类的一个例子而已。我们的任务是将数据框转化为 list，再将 list 转化为矩阵，然后将矩阵转化为数据框还原的过程，在将数据框转化为 list 之前，通过 `str` 函数我们看到数据框 `iris` 中有一列数据是以因子的样式记录的，所以需要将其转化为字符。原因很简单，之前说过因子是以整数记录的整数关联到因子水平，我们担心在操作过程中丢失因子水平，仅剩下整数就不好办，所以这里使用了 `as.character` 函数将 `iris$Species` 这一列从因子形式转化为字符，并重新赋值给 `iris$Species`，这样重新使用 `str` 函数查看 `iris$Species` 的数据类型就变成了 chr；`unclass` 函数用于解散一个类，就是将类解散为组成元素，数据框是由等长的向量组成的，`unclass` 之后这些向量降级组成了一个 list，list 的每一个元素是原来数据框的。

第 6 行代码再次查看 `temp` 的结构，发现 `temp` 变成了一个包含 5 个元素的 list；第 7 行代码第一次使用了一个相对难以理解的函数，`do.call` 函数是一个非常有用而又一言半语难以说清的函数。

首先这个函数是一个调用函数，请注意调用分为两部分，即 `call`（调取）和 `do`（执行）部分，不仅调还要用。其次当被调用的函数的参数无限多时，就需要一个 list 来存储这些参数，比如 `cbind(Sepal.Length, Sepal.Width, Petal.Length, .....)`，你可能想着将很多个诸如 `Sepal.Length`、`Sepal.Width`、`Petal.Length` 等长向量传递给 `cbind` 按列捆绑在一起，但是传参时又不想烦琐地赋值，那就将 `Sepal.Length`、`Sepal.Width`、`Petal.Length` 组成一个 list 吧，即 `temp`，一起传给 `cbind`，但是 `cbind` 是不接受 list 作为参数的，这时就用到了 `do.call`，它可以将一个函数的所有参数作为一个 list 传递给被调用的函数，这里 `do.call` 就将 `temp` 作为参数集传递给了 `cbind`，因为在 R 里，参数如果使用等号赋值，就会按照先后顺序赋值。你可能会想到既然参数可以作为 list 传参，那么函数能不能作为 list 批量调用呢？能，这个后面会讲到。

第 8 行代码将 `temp` 转化为数据框；第 9 行代码查看数据结构发现原来类型为数值的变量都变成了字符，当然需要改回来；第 10 行代码可以使用 `as.numeric` 函数将字符变量转化为数值，其他的几



个我们就不举例了，你可以使用 `as.numeric` 函数将其转化为数；第 11 行代码将 `Species` 列由字符转化为原来的因子变量，使用的是 `as.factor` 函数，其实这些是可以批量转化的，不过限于此处是本书的基础部分，我们后面再讲。

- 对象转化：数据框—list—向量—矩阵—数据框

```
1 iris$Species <- as.character(iris$Species)
2 numcols <- ncol(iris)
3 namecols <- names(iris)
4 temp <- unclass(iris) # 数据框转化为 list
5 temp <- unlist(temp)
6 str(temp)
7 temp <- matrix(temp, ncol = numcols, byrow = F) # 向量转化为矩阵
8 temp <- data.frame(temp, stringsAsFactors = F) # 矩阵转为数据框
9 names(temp) <- namecols
10 str(temp)
11 temp$Sepal.Length <- as.numeric(temp$Sepal.Length)
12 temp$Species <- as.factor(temp$Species)
```

上述代码中，第 1 行代码将因子变量转化为字符；第 2 行代码使用 `ncol` 计算数据框列数，并赋值存储备用；第 3 行代码使用 `names` 函数获得列名称，并赋值存储备用；第 4 行代码将数据框 `unclass` 为 `list`；第 5 行代码使用 `unlist` 函数将 `list` 解散为向量；第 6 行代码使用 `str` 函数查看数据结构发现 `temp` 已经变成了 `chr` 组成的向量；第 7 行代码使用 `matrix` 函数将向量转化为矩阵，矩阵的 `ncol` 列数指定为原数据框的列数，并指定 `byrow` 参数是否按行排列，你可以将 `F` 修改为 `T`，以检查按行和按列排列的区别；第 8 行代码将矩阵转化为数据框；第 9 行代码将原来的列名重新赋值给 `temp` 的列名，这里用到的是 `names` 函数，当然也有很多其他函数可以参考。如使用 `colnames`、`rename`（`dplyr` 包）进行修改列名称；第 10 行代码查看数据结构；第 11 行和第 12 行代码修改变量的数据类型。

- 分组转化：数据框—list

```
1 temp <- split(iris, iris$Species) # 将数据框分组
2 str(temp)
3 temp <- do.call("rbind", temp)
4 str(temp)
```

在数据处理过程中，我们经常在数据框和 `list` 之间转化，通常包括两种情况：将数据框按因子分组为 `list` 和将 `list` 合并成行为数据框，后面这种在数据处理时会经常用到。上述代码中，第 1 行代码将数据框 `iris` 按照 `Species` 分组成为多个小的数据框，分组变量 `Species` 列不一定要在数据框 `iris` 中，但是一定要与 `iris` 等长；第 2 行代码查看 `temp` 的结构，发现 `temp` 变成了一个 `list`，`list` 包含 3 个元素，是 3 个结构完全相同的数据框；第 3 行代码将元素结构相同的 `list` 捆绑为数据框，也是使用 `do.call` 调用 `rbind` 函数将 `temp` 作为 `rbind` 的参数打包传递，这里将函数名 `rbind` 的外面包括一个英文引号，即 `do.call(c("rbind"), temp)`，换句话说，`do.call` 也可以批量调取函数，因为第 1 个参数既然可以接受一个包含函数名的向量，就可以再给这个向量添加多个函数名，如 `c("rbind", "cbind", "split")` 等，至于具体事例我们后面再进行讲述。

## 1.3 R 语言的重器：函数

重剑无锋，大巧不工，函数可以说是 R 语言的“国之重器”，很多高级语言都是通过函数执行的。所谓函数就是具有一定输入和输出的代码块，在 R 语言里没有函数就寸步难行，所以我们需要了解一些有关函数的基础知识，不仅 R 语言需要这些基础知识，学习其他语言也要关注这些基本属性。

### 1.3.1 自编函数

所谓自编函数就是使用者自己编制的函数，尽管自编函数在 R 里面很少使用，但是有时候为了调用方便，经常将一个代码块编织成一个函数，通过调用来减少代码量，代码量少不是为了说明代码写得好，而是为了方便。请大家记住，在数据处理的过程中，为实现同一个目标，对代码的编写要求就像火云邪神的说法一样：天下武功，唯快不破。在 R 里编制函数是要遵守一定的范式的，如下所示。

```
myfun <- function(参数 1, 参数 2, ...) 主体
```

函数一般有一个名称，通过赋值的方式将函数传递给函数名称，函数以 `function` 开头，后跟一个英文括号，用于指定参数在后面是函数的主题，如果函数主题只有一行代码则不用花括号，如果是一行以上，则需要使用花括号将代码块括起来。

- 函数示例

```
1 myfun <- function(a, b) a + b
2 myfun(a = 2, b = 3)
3 myfun2 <- function(a, b) {
4   y = a + b
5   y*a
6 }
7 myfun2(a = 2, b = 3)
```

上述代码中，第 1 行代码创建一个叫 `myfun` 的函数，函数包括两个参数，主体部分使用一行代码对参数的值求和；第 2 行代码测试函数，发现能够正常执行；第 3 行代码起创建了另外一个自编函数，函数包括两个参数，函数主体包括两行代码，所以使用花括号括起来，代码之间直接换行不用使用任何分隔符；最后一行代码测试自编函数能否正常使用。

有时候在函数里面的处理动作不止一个，可能有很多中间操作，这必将产生很多中间结果，我们可能需要它们中的多个结果，而 R 里的函数默认情况下仅仅返回最后一行代码产生的结果，这时就需要指定返回的结果，如下所示。

- `return` 设定返回值

```
1 myfun3 <- function(a, b) {
2   y = a + b
3   m = y*a
4   return(list(y, m))
}
```

```
5 }
6 resultls <- myfun3(a = 2, b = 3)
```

上述代码中，在 myfun3 的末尾添加一个返回函数 return，用于指定需要返回的对象。如果需要返回多个对象，则需要将对象捆绑为一个对象，这里使用 list 函数将 y 和 m 捆绑成了一个 list 返回，捆绑为 list 返回是最简单最方便的多结果返回方式，返回之后可以按照元素的索引从 list 中提取返回结果。

除了结果外，还有必要谈一谈函数的参数，有关参数需要了解两点：参数的默认值和参数的顺序。

- 参数的默认值

```
1 myfun3 <- function(a, b = 4) {
2   y = a + b
3   m = y*a
4   return(list(y, m))
5 }
6 myfun3(a = 2)
7 myfun3(a = 2, b = 6)
8 myfun3 <- function(a, b) {
9   y = a + b
10  m = y*a
11  return(list(y, m))
12 }
13 myfun3(a = 2)
# Error in myfun3(a = 2) : argument "b" is missing, with no default
```

上述代码中，第 1 次将函数 myfun3 的第 2 个参数 b 的默认值设置为 4（默认值是重新指定的），第 1 次调用函数 myfun3 仅传了一个参数，函数能正常执行返回结果；第 2 次调用函数修改了函数的默认值重新传入了一个新的数值，函数仍能正常执行；但是我们将函数的默认值去掉之后，重新调用传参，仅传入一个参数时，函数就不能正常执行，报错了，通过上面的测试不难发现设置有默认的参数不必传参，但是没有指定某默认值的参数必须在传参后才能保证函数正常执行。

- 参数的顺序

```
1 myfun <- function(a, b) {
2   x <- paste("参数 a 的值: ", a)
3   print(x)
4   y <- paste("参数 b 的值: ", b)
5   print(y)
6 }
7 myfun(a = 2, b = 5)
8 myfun(2,5)
9 m <- 5
10 n <- 3
11 myfun(m,n)
12 myfun2 <- function(..., b = 2) {
```

```
13 a <- list(...)
14 a <- unlist(a)
15 print(a + b)
16 }
17 myfun2(1,2,n,m, b = 3)
```

一般函数需要使用“=”号传参，这也是一个非常好的习惯，但是有时候分析员时间紧迫，为了少敲几次键盘，而不再使用等号传参，如 `myfun(2,5)`。可以看到，其实 R 里面函数的参数，如果没有指定传参，则会按照顺序赋值，比如第 1 个值传给第 1 个参数，第 2 个值传递给第 2 个参数，依次类推；稍微将 `myfun` 改造一下就成了 `myfun2`，`myfun2` 的参数，“...”表示在参数 `b` 之前可以有很多个参数，函数主体内将这些参数统一转化为一个 `list`，然后赋值给 `a`，将 `a` 解散为向量，最后打印 `a+b` 的结果；最后一行代码我们调用了 `myfun2`，在参数 `b` 之前向它传递了 4 个参数。

在 `myfun` 函数内部我们使用了 `paste` 函数，其作用是将多个对象粘贴为一个字符串输出，但是粘贴时有分隔符的要求，`paste` 函数用于指定分隔符的参数有两个：`sep` 和 `collapse`，它们是有明显区别的，如下所示。

- `paste` 参数 `sep` 和 `collapse` 的区别

```
1 x <- c("a", "b", "c", "d")
2 y <- c("e", "f", "g", "h")
3 paste(x, y, sep="+")
4 paste(x, y, sep="+", collapse=",")
5 paste(x, y, sep=",", collapse="+")
6 paste(x, sep = ",")
7 paste(x, collapse = ",") # 将向量的所有元素按照字符分隔
```

上述代码中第 1 行和第 2 行代码创建了两个向量 `x`、`y`；第 3 行代码将 `x`、`y` 对应的元素粘贴在一起，并使用 `sep` 指定分隔符为“+”，可以看到结果是一个和 `x`、`y` 等长的向量，也是包含 4 个元素；第 4 行代码指定了 `collapse` 的参数为“，”，发现粘贴后的结果变成了一个长的字符串（string），合并时使用 `collapse` 指定的符号分隔，可见 `collapse` 是指在对象元素粘贴的基础上再将其结果粘在一起；第 4 行和第 5 行代码显示了 `sep` 和 `collapse` 的区别；如果我们仅仅传递一个向量给 `paste` 函数，同时指定 `sep` 的分隔符为“，”，其结果会发生什么呢？第 6 行代码的结果表明输出的结果和 `x` 没有区别，这表示 `paste` 函数仅有一个参数是默认该参数的元素与空值粘贴；那如果我们想将向量的每个元素粘贴在一起呢？没错，通过 `collapse` 指定分隔符即可（见第 7 行代码）。

另外需要注意的是，函数有其自身的环境。软件有软件的环境，R 也有其自身的环境，你所创建的对象包括数据对象、函数等都是其组成元素，其实函数圈定了一个输入输出的环境，一般情况下你输入什么参数，函数就只能使用什么参数，但既然有了一般怎么能缺少特殊！

- 环境（environment）

```
1 rm(list = ls())
2 a <- 3
3 ls()
4 myfun <- function(x) {a + x}
```

```

5 myfun(4)
6 myfun2 <- function(x) {
7   a <- a + x
8   a + 1
9 }
10 myfun2(4)
11 a

```

上述代码中，第 1 行代码移除内存中所有的对象，ls 函数用于获取内存中已经加载的对象名称，然后将这些名称传递给 rm 函数的参数 list，rm 就将环境中的所有对象移除了；第 2 行代码创建一个新的对象；第 3 行代码查看内存中的对象名称，这时仅存在一个刚刚创建的对象 a；第 4 行代码创建了一个自编函数，自编函数只有一个参数 x，但主体使用了 a 这个对象，按道理函数在其自身圈定的范围内是无法找到对象 a 的，因为既没有传参也没有在函数内部创建对象 a，但是这时特殊情况出现了，R 允许函数先在函数自身的环境内按名称搜索对象，如果找到了就是用自身环境内的对象，如果搜索不到则扩大搜索范围，允许其在 R 的环境内搜索对象；第 5 行代码可以看到 myfun 正常执行并使用了自身环境之外的 R 对象 a；那问题来了，既然是 R 允许函数扩大所搜范围，那如果在函数内部对 R 环境中的同名对象进行修改会不会改变 R 环境中的对象呢？第 6 行代码起重新创建了一个自编函数 myfun2，在其内部使用了 R 环境中的对象 a，并将其加上参数 x 后重新赋值给了 a，然后再对 a 加 1；倒数第 2 行代码我们调用了函数，并将参数 x 赋值为 4，结果发现函数使用了 R 环境中的对象 a，并在其自身环境中创建了一个同名的对象 a，其返回的结果正好是其自身环境中的 a 加上 1；而 R 环境中的 a 有没有改变呢？最后一行代码证明 a 没发生变化，说明 R 允许函数扩大对象的使用范围，却不允许函数创建和修改的对象扩大到 R 环境，必须以结果的形式返回并赋值才行。

### 1.3.2 有用的 R 字符串函数

R 语言中有很多非常有用的函数，我们在之前也讲了很多有用的函数，但都是比较简单的函数（简单但是常用），下面主要介绍一下和正则表达式相关的函数，主要用于处理一些字符类的函数。

正则表达式（Regular Expression）是一种匹配检索模式，用于查询符合一定规则的字符，想了解更多可以百度搜索“正则表达式 30 分钟入门教程”，这个教程非常简单，但是正则表达式实在是“天书”，如果不是专门做这个，了解一些基础的概念和常见的写法就可以了，比如什么叫元字符、什么叫转义等。

首先经常碰到字符分裂类的问题，一般分为两种情况：按照固定字符分裂和按照固定长度分裂。前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂。

- 字符串分裂：按字符分裂

```

1 mystring <- "前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂"
2 strsplit(mystring, split = "字符", perl = T) # 固定字符

```

```
3 strsplit(mystring, split = "字符|进行", perl = T)
4 mystring <- c("前者表示指定一个分裂字符串", "当字符串中出现该固定模式时就进行分裂", "后者表示每隔多少个字符进行分裂")
5 strsplit(mystring, split = "字符|进行", perl = T)
```

上述代码中，第 1 行代码创建一个字符串对象；第 2 行代码使用基础的 `strsplit` 函数指定字符切分，第 1 个参数指定被切分的对象，第 2 个参数指定切分的字符，`perl` 参数指定正则表达式是否与 `perl` 语言兼容，其实 `split` 用来指定一个正则表达式，只不过这里的模式比较简单，就是按字符切分；第 3 行代码使用了一个正则表达式中“或”模式，指定当碰到“字符”或“进行”时进行切分，“|”在正则表达式或 R 中都是“或”的意思；

- 字符串分裂：按固定长度分裂

```
1 mystring <- "前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂"
2 strsplit(mystring, split = "(?<=. {3})", perl = T)
3 substr(mystring, start = 1, stop = 3)
4 startpoint <- seq(1, nchar(mystring)-1, 3)
5 stoppoint <- seq(2, nchar(mystring), 3)
6 substring(mystring, startpoint, stoppoint)
```

上述代码中，第 2 行代码使用了一个比较难懂的正则表达式，“`(?<=. {3})`”表示匹配任意 3 个字符后面的字符，外面的英文小括号表示分组提取，这样 `split` 函数就有了分裂的依据，然后就可以得到按 3 个字符长度分裂的结果，当然你也可以改成任意一个字符长度，如果看不懂，不要急，下面提供另外一种方法；第 3 行代码使用 `substr` 函数获取指定位置的字符，第 2 个参数指定字符的起始位置，第 3 个参数指定字符的结束位置，整体来说提取字符串的第 1 个到第 3 个字符（包括第 1 和第 3 个），我们将 `substr` 稍加改造就可以用于固定长度分隔；第 4 行代码中 `seq` 生成一个等差数列向量，向量从 1 开始，到字符串 `mystring` 的字符数减去 1 结束，`nchar` 用于获取字符串的字符数，数列的公差是 3，也就是说每隔 3 个字符；第 5 行代码生成另外一个等差数列，数列的起始位置为上一个数列的公差，这里是准备按照 3 个字符分隔，所以其实位置为 3，结束位置为字符串的字符数，公差仍然是 2；第 6 行代码将 `startpoint` 和 `stoppoint` 赋值给函数 `substr` 的参数即可完成按 3 个字符长度的分隔。

- 字符串的查找替换

```
1 mystring <- "前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂"
2 grepl("字符", mystring, perl = T)
3 gsub("字符", "b", mystring, perl = T)
```

经常需要查看一个字符串中是否包含某种模式的字符，比如上述代码中的第 2 行代码使用 `grepl` 函数搜索字符串中是否包含“字符”，`grep` 是 `global search regular expression` 的缩写，而 `l` 表示 `logic`，也就是说 `grepl` 返回的是一个逻辑值；第 3 行代码的 `gsub` 函数用于查找某种模式并替换另外一种模式，这里将“字符”替换为“b”。

在数据抓取的任务中我们经常需要做数据提取，但通常又不知道模式的起始范围，那就只能用函数指定一个模式去查找模式的起始位置，`regexr` 函数可以完成这样的任务，如下所示。

- 模式查询函数

```
1 mystring <- "前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂"
2 y <- regexpr("前者.*后者", mystring, perl = TRUE)
3 y
4 z <- y + attr(y, which = "match.length") - 1
6 substr(mystring, start = y, stop = z)
```

regexpr 函数用于查找模式的起始位置和模式的长度。比如正则表达式“前者.\*后者”表示以“前者”开始且以“后者”结束，中间任意多个任意字符的模式。regexpr 函数的第 1 个参数用于指定正则表达式模式，第 2 个参数指定数据，第 3 个参数用于设置时兼容 perl 模式，regexpr 函数返回的是模式之前有多少个字符，并且以属性 (attr) 的方式返回模式本身有多长 (match.length)，使用 attr 函数可以提取向量的属性，这样我们有了模式的起始位置加上模式的长度就很容易算出模式的结束位置，有了起始和结束位置就可以提取模式了。

上述代码中，第 2 行代码查询模式；第 3 行代码查看查询结果，看到结果有一个叫作 match.length 的属性；第 4 行代码计算结束位置，attr 函数用于提取属性，因为对象可能有多个属性，所以 which 参数用于指定属性的名称；第 5 行代码就可以按照起始和结束位置提取字符了。

- 模式查询函数：多次匹配

```
1 mystring <- "前者表示指定一个分裂字符串，当字符串中出现该固定模式时就进行分裂；后者表示每隔多少个字符进行分裂"
2 m <- gregexpr("字符", mystring, perl=TRUE)
3 regmatches(mystring, m)
```

regexpr 函数仅能匹配第一个出现的模式，但有时候要匹配出字符串中所有符合的模式，这里就需要 gregexpr 函数出场了，它和 regexpr 函数返回的结果一样，只不过它匹配多次，另外为了替换 substr 函数，上述代码中使用了 regmatches 函数，用于提取匹配到的模式，它会根据 gregexpr 函数的结果自动计算模式的起止位置，然后提取模式。

想了解更多关于字符和正则表达式的函数，可以输入“?gsub”查看帮助文档，你可以看到和正则表达式相关的很多函数，不必每个都记住，但是需要记住它们都有相同的参数，第 1 个参数是指定模式，第 2 个参数 x 指定数据，x 可以是单个字符串，也可以是一个字符串向量，换句话说这些函数都能进行向量计算，一次批量处理多个字符串，同时也推荐使用 stringr 包，一个专门用来做字符处理的 R 包。

## 1.4 控制流在 R 语言里只是一种辅助工具

控制流就是在一个动作中所做的控制操作，如循环、判断、跳错等操作。在向量计算的背景下很少用到控制流，比如可能不需要循环就能将一个向量的数据全部计算一遍（上面已有简单举例），即便是判断也可以做向量化处理，所以控制流在 R 里面算是不太常用的工具，但是在处理比较大的

原始数据时，可能会引起你对控制流的思念，特别是处理非常脏乱的数据时，所以掌握 R 中的控制流能够帮助你达到事半功倍的效果。

### 1.4.1 判断

判断语句应该是编程中比较常见的现象，经常需要判断一种状态然后做出下一步执行的决定，在 R 中判断语句的结构也很简单，如下所示。

- 判断语句：(1)

```
1 a <- 2
2 b <- 3
3 if (a < b) b
4 if (a < b) {
5   b
6 }
```

最常见的是判断一个条件是否成立，如果成立则执行一条语句或者一个代码块，比如上述代码是判断 a 是否小于 b，如果小于则输出 b，如果不小于则什么也不做，什么也不做就省略不写了。这应该是最简单的判断语句，当然如果你要执行一个代码块（多语句）请添加花括号。

我们有时候会根据一个条件做出两种决定，比如如果条件 a 发生，将执行什么，如果条件 a 不发生则执行什么，如下所示。

- 判断语句：(2)

```
1 if (a < b) a - 1 else b
2 if (a < b) a - 1
3 else b
```

需要注意的是，在 R 里面的 else 语句或 else if 语句必须紧跟上一个判断结果的代码块，否则会报错，如果上述代码中我们将 else 断开上一句判断结果的代码块，再执行就会报错。

偶尔还会进行多重判断，这就需要更加复杂的判断语句，如下所示。

- 判断语句：(3)

```
1 if (b > 4) {
2   print("b > 4")
3   print(paste("b 的值为", b))
4 } else if (b > 3) {
5   print("b 不大于 4")
6   print("但是 b 大于 3")
7 } else {
8   print("b 不大于 3")
9   b
10 }
```

上述代码中，if 语句先进行了第一次判断，然后 else if 进行了第二次判断，最后 else 根据判断结果分析第三种情况；很简单，每一种情况都有处理的代码块，用花括号括起来，还是那句如果 else 或 else if 和上一层的代码块断开，则会报错，一定要保证 if-else 的连贯性，如下所示。



```

1 if (b > 4) {
2   print("b > 4")
3   print(paste("b 的值为", b))
4 } else if (b > 3) {
5   print("b 不大于 4")
6   print("但是 b 大于 3")
7 }
8 else {
9   print("b 不大于 3")
10  b
11 }

```

条件判断在 R 里面还有一个特点,就是不能进行向量计算,换句话说这种判断是针对数据点的,而不能批量进行,除非和循环一起使用,否则它仅仅使用第一个元素进行判断,如下所示。

- R 的判断不能向量化

```

1 a <- 1
2 b <- 3:7
3 if (a < b) b
4 b <- 0:7
5 if (a < b) b
6 ifelse(a < b, b, a)
7 ifelse(a < b, b - 1, a + 1)

```

上述代码中,可以看到,当用 a 与向量 b 比较时, R 仅仅使用了 b 中的第 1 个元素和 a 对比,如果第 1 个元素大于 a 就会输出整个 b,否则就不会输出任何东西,我们将 b 的一个元素改成分别大于 a 或者小于 a 的情况就可以看出 R 是怎么处理的了,当然如果要使用向量计算也不是没有办法,ifelse 就有这种功能,它的第 1 个参数用于指定判断的条件,如果判断条件成立,则输出第 2 个参数,如果不成立则输出第 3 个参数,倒数第 2 行代码,如果向量 a 的元素小于向量 b 中对应的元素则输出 b 中的元素,否则输出 a 中的元素;最后一行代码的意思是如果 a 的元素小于 b 中相应的元素,则输出 b 中相应元素减去 1 的值,否则输出 a 中相应元素与 1 的和。

判断语句

- if(条件) 语句
- if(条件) 语句 else 语句

大家只要记住判断语句的结构就可以了,缩略记就是 if()。

## 1.4.2 循环

另外一种控制流语句就是循环,在 R 里面几乎很难看到循环,因为绝大多数循环都可以使用向量计算的方式解决掉,仅在数据抓取时才使用循环,其目的是降低抓取效率,防止被对方封停 IP 地址,其实循环很简单,人们使用循环的目的是使思路简单,而向量计算思路实现起来就比较困难。

### for 循环结构

- for (变量 in 清单) 语句
- for (变量 in 清单) {  
    语句 1  
    语句 2  
}

在 R 中 for 循环最常见，其结构就是 for()，最容易记住，括号里指定一个变量，将清单中的值依次取出赋值给变量，然后执行下面的语句，直到清单中的元素全部取一遍为止，如下所示。

- for 循环

```
1 seq(1, 9, 2)
2 for (i in seq(1, 9, 2)) print(i + 1)
3 for (i in seq(1, 9, 2)) {
4   print(i + 1)
5   print(i - 1)
6 }
```

上述代码中，第 1 行代码的 seq 函数用于生成等差数列，第 1 个参数用于指定数列的起点，第 2 个参数用于指定参数的终点，第 3 个参数用于指定公差；第 2 行代码开始一个 for 循环，将序列中的元素一一取出开始循环，循环的部分为打印元素加 1 之后的结果；第 3 行代码起开始一个 for 循环，循环部分是一个代码块，打印元素 i 加 1 和减 1 的结果，当清单中的元素执行一遍后，循环自动终止。

我们将上面的 for 循环稍加改造添加了一条判断语句，仍然是从序列中取出元素 i，如果元素 i 小于 3，则执行 next 函数，表示什么都不做，直接进行下一个元素循环，如果不小于 3 则打印 i 加 1 的结果。如下所示。

- for 循环与条件判断

```
1 for (i in seq(1, 9, 2)) {
2   if (i < 3 ) next else {
3     print(i + 1)}
4 }
```

其实除了 for 循环外，还可以使用另外一种循环形式，即 while 循环，其结构如下。

### while 循环

- while (条件 1) 语句

其使用方法很简单，几乎和 if 语句一样，就是 if 语句的重复形式，举个例子，如下所示。

- while 循环

```
1 a <- 2
2 while (a < 30) {
3   print(a)
4   a <- a + 4
5 }
```

上述代码中，第1行代码创建一个对象，并赋值为a；第2行代码起开始一个循环判断，如果a小于30，则输出a的值，同时将a加上4再复制给a，循环直到a的值不小于30为止。

其实，在R里面绝大多数的判断和循环都是没必要的，只有极少情况下才会使用控制流，而使用控制流的目的已经不再是为了常见的判断和循环思维逻辑，可能是为了节省时间，比如你有1亿条文本要清洗处理分词，如果用向量计算，那用分词函数一次就可以了，但是1亿条还是要花费一段时间的（并行和分布除外）。那么问题来了，如果使用向量计算，中间突然出现意外报错，是不会返回结果的，这样又要从头再来，而使用循环加向量计算的方式可以返回部分已经做好的工作，如果中间报错，下次直接从报错的那部分开始就可以了。这一部分我们将在代码优化和思维优化的章节进行详解。

## 1.5 数据的读入与输出

数据的输入基本上是一个项目的开始，而项目在结尾处总要输出结果，可以说数据的输入就是将外部数据导入到R环境，数据的输出就是将R环境的对象导出到R环境之外，可想而知多么重要，我们在这里仅仅讲述读入CSV、TXT文件和数据库等几种方式，至于其他格式并不常见，无论是其他环境还是其他语言基本上都是通过这些基本数据格式之间沟通，如果非得弄一些稀奇古怪的数据格式只能说明还没有真正掌握R语言丰富的内容，因为高手总希望获得最基础最通用的沟通方式。

### 1.5.1 常见数据格式的输入 / 输出（CSV、TXT、RDATA、XLSX）

CSV、TXT格式基本上是大多数语言支持的数据输入方式，甚至数据库入库出库也是这两种常见的格式，所以这才是通用的东西。CSV就是Comma-Separated Values，即以逗号分隔的数据表，所谓逗号分隔就是列和列之间以英文的逗号分隔。另外所有的Excel文件都可以通过“另存为”的方式保存为CSV格式，然后在R语言中读取CSV，如下所示。

- 读取CSV文件

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第1章万事不只开头难/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)
2 tail(hospital)
3 str(hospital)
```

上述代码中，第1行代码通过read.csv函数读取CSV文件，第1个参数指定文件的完整路径，这里我们读取了全国两万多家医院的规模数据，header参数用于指定数据第1行是否是列名称，为真表示文件第1行是列名称，否则R会将文件第1行当作数据然后默认给数据的列命名，sep参数用于指定分隔符，这里使用英文逗号分隔，stringsAsFactors参数用于设定是否将字符型变量转化为因子，这一点很重要，如果数据量不是很大请设置为F，如果数据量很大且转化为因子后因子水平比较少请设置为T，这样可以节省数据占用的内存；第2行代码通过tail函数查看数据的最后6行；第

## 探寻数据背后的逻辑：R 语言数据挖掘之道

3 行代码查看数据结构，发现 read.csv 函数读取数据后将数据转化为数据框。

- 读取 TXT 文件

```
1 hospital <- read.table('H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 1 章万事不只  
开头难/data/hospital.txt', header = T, sep = "\t", stringsAsFactors = F)  
2 tail(hospital)  
3 str(hospital)
```

上述代码中，第 1 行代码通过 read.table 函数读取 TXT 或者无后缀文件，第 1 个参数指定文件的完整路径，header 参数用于指定第 1 行是否是列名称，sep 参数用于指定分隔符，这里使用制表符分隔，请记住 TXT 文件的分隔符多种多样，比如“;”、“#”和空格等，而且不同的环境还不一样，比如在 Hive 中导出的一般是无后缀的文件分隔符，为“\001”等，stringsAsFactors 参数用于设定是否将字符型变量转化为因子；第 2 行代码通过 tail 函数查看数据的最后 6 行；第 3 行代码查看数据结构，read.table 函数读取数据后将数据转化为数据框。

- 输出 CSV、TXT 文件

```
1 write.csv(hospital, file = ('H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 1 章万  
事不只开头难/plot/hospital.csv', row.names = F)  
2 write.table(hospital, file = "E:/业余/zimeiti/探寻数据背后的逻辑：R 语言数据  
挖掘之道/bookwriting/第 1 章万事不只开头难/plot/hospital.txt", row.names = F)
```

上述代码中，第 1 行代码使用 write.csv 函数输出 CSV 格式的文件，即将文件写入硬盘，第 1 个参数用于指定输出的对象，对象一般是数据框格式，file 参数用于指定输出的完成路径，row.names 参数用来指定是不是输出行名称，默认是用输出行编号的，所以一般需要特别指出不要输出行编号，否则行编号就会变成一列新的数据；第 2 行代码使用 write.table 函数输出 TXT 格式的文件，参数和 write.csv 相同。

- 输入 / 输出 RData

```
1 getwd()  
2 setwd("D:/BF/Documents")  
3 a <- 1:3  
4 b <- c("a", "b", "b")  
5 alist <- list(a, b)  
6 save(hospital, alist, file = "temp.RData")  
7 rm(list = ls())  
8 load("temp.RData")  
9 ls()
```

RData 是 R 语言特有的一种数据存储格式，它能存储所有 R 语言环境中的对象，也就是说它不限制对象类型。上述代码中，第 1 行代码的 getwd 函数用于获得当前工作目录，如果输入输出文件没有指定具体的路径，R 就会在当前工作目录下搜索文件读取或输出，Windows 系统默认为文档目录；第 2 行代码的 setwd 函数用于设定新的工作目录；第 3~5 行代码创建了一个 alist 对象；第 6 行代码将数据框 hospital、表单 alist 一起保存在 temp 的 RData 中，这里我们没有指定完整的路径，所以默认会保存到当前工作目录；第 7 行代码移除内存中所有的对象，可以看到 Rstudio 中的

environment 中已经没有了对象；第 8 行代码重新将名为 temp 的 RData 加载到内存，没有指定完整路径 R 就会去当前目录下查找；第 9 行代码的 ls 函数用于查看内存中的 R 对象，可以看到仅仅有刚刚加载的 hospital 和 alist；其实 RData 主要是用于保存格式比较特殊的 R 对象，比如构建完成的各类模型等。

在一些市场研究公司，case by case（具体问题具体分析）的项目比较多，数据一般是以 Excel 读取的格式 XLSX 存储的，很少有以数据库、CSV 或 TXT 格式存储的，这种情况下，一种方式是将 XLSX 格式的文件存储为 CSV 或 TXT 读取，另外一种方式是直接读取 XLSX 文件。如下所示。

- 输入 XLSX 文件

```
1 library(xlsx)
2 temp <- read.xlsx('H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 1 章万事不只开头难
/data/hospital.xlsx', sheetIndex = 1, as.data.frame = T, header = T, encoding
= "UTF-8")
3 head(temp)
```

xlsx 包用于处理 XLSX 文件，你可以通过前面已经讲过的安装方式安装这个包，其中 read.xlsx 函数读取 XLSX 文件，第 1 个参数用于指定文件的路径，sheetIndex 参数用于指定读取第几个 Sheet，也可以通过 sheetName 参数设定读取的 sheet 名称，as.data.frame 用于指定是否将数据转化为数据框，header 参数和上面函数中的意义一样，即是否将数据第 1 行设为列名称，encoding 用于指定文件的编码，我们将在本章的 1.5.3 节中详解，这个函数有个缺点，速度非常慢，慢到无法忍受，2 万条数据要好几分钟。

以上我们讲解了各种文件的输入和输出方法，这样第 1 章的 R 基础部分基本上可以过关了，也就是说你对 R 语言的基础内容已经掌握得比较全面了。

## 1.5.2 数据库连接：Oracle、MySQL 及 Hive

本节内容将基本上脱离基础知识，可能需要有点其他领域的积累才能看懂或者阅读全书后才能彻底明白，我建议读者先将此节内容搁置，因为这一节真的没有标准答案：我的系统能搞定的流程和代码放到你的系统上不一定能搞定，牵涉甚广，但是我们这里尽量讲解一些通用的内容，这并不是说本节的内容不重要，因为一旦数据上升到平台，就需要和数据库打交道，即从数据库中读取数据，然后将结果写入数据库。

这里使用 RJDBC 与数据库沟通，因为它对多种数据均适用，所以这里暂时以这种简单的方式与数据库沟通，但是在使用 RJDBC 之前首先要安装 Java 并配置环境变量，请参看第 10 章，安装完成之后可以在 R 里面安装 rjava 和 RJDBC 包，如果还是不懂，就需要通过上网搜索了，比如说如何安装 Oracle 客户端、如何配置 Java 的环境变量等。

- RJDBC 与 Oracle 数据库

```
1 install.packages("rjava")
2 install.packages("RJDBC")
3 library(RJDBC)
```

```
4 jdbcDriver <- JDBC(driverClass = "oracle.jdbc.driver.OracleDriver",
classPath = "c:/ojdbc6.jar", " ")
5 conn <- dbConnect(jdbcDriver, "jdbc:oracle:thin:@//172.35.28.75:1521/bigdata",
"well", "well12")
```

上述代码中，包加载完成后需要使用 JDBC 函数指定一个 jdbc 驱动，第 1 个参数用于指定驱动的类型，这里这个版本连接的 Oracle 就是指定 Oracle 类型，参数 classPath 用于指定驱动存放的路径，我的 jdk 是 1.6 版本所以选择的驱动也是 1.6 版本，你可以去百度搜索并下载相应的 ojdbc6.jar 文件，不一定是这个版本，但尽量要和你的 jdk 版本一致；dbConnect 函数用于与数据库建立连接，第 1 个参数指定上一步已经构建的驱动文件，第 2 个参数用于指定数据库连接的 ip，172.35.28.75 指所连接远程数据的 ip，1521 指 Oracle 的端口，一般都是 1521，后面的 bigdata 指库的名称，因人而异，这里需要注意的是你要保证本机具有访问远程数据库的权限，如果没有请找运维的同事帮忙开通，而且需要注意这个参数还有另外一种写法“jdbc:oracle:thin:@172.35.28.75:1521/bigdata”，第 3 个参数用于指定用户名，第 4 个参数用于指定用户密码，通过以上的连接基本上可以和数据库沟通了。

### • RJDBC 与 MySQL 数据库

```
1 library(RJDBC)
2 jdbcDriver <- JDBC(driverClass="com.mysql.jdbc.Driver", classPath="c:/mysql-connector-java-5.1.18-bin.jar", " ")
3 conn <- dbConnect(jdbcDriver, "jdbc:mysql://172.35.28.75:3306/bigdata?
useUnicode=true&characterEncoding=UTF8", "mysql", "mysql_2016")
```

连接 MySQL 与连接 Oracle 的方法基本相同，上述代码中，使用 JDBC 函数指定一个 jdbc 驱动，第 1 个参数用于指定驱动的类型，这里指定 MySQL 类型，参数 classPath 用于指定驱动的存放路径，你可以去百度搜索并下载相应的“mysql-connector-java-5.1.18-bin.jar”文件；dbConnect 函数的第 1 个参数指定上一步已经构建的驱动文件，第 2 个参数用于指定数据库连接的 ip，172.35.28.75 指所连接远程数据的 ip，3306 指 MySQL 的端口，一般都是 3306，后面的 bigdata 指库的名称，因人而异，再后面的“?useUnicode=true&characterEncoding=UTF8”指定数据的编码方式，因为 MySQL 数据库用的是 gbk 编码，而项目中我们一般使用 utf-8 编码数据，所以要告诉数据库我们使用的编码方式，不然上传的数据会出现乱码现象，后两个参数用于指定用户及密码。

连上了数据库就需要学习其基本的操作，以下可能涉及一些简单的数据库语言：SQL，请参阅其他资料学习 SQL 相关知识。

### • RJDBC 常见的操作

```
1 temp <- data.frame(a = 1:4)
2 dbWriteTable(conn, "TESTJDBC", temp)
3 dbWriteTable(conn, "TEMP", temp)
4 dbSendUpdate(conn, "insert into TESTJDBC select * from TEMP")
5 dbSendUpdate(conn, "drop table TEMP")
6 temp <- dbGetQuery(conn, "select * from TESTJDBC")
7 dbSendUpdate(conn, "truncate table TESTJDBC")
```

上述代码中，第 1 行代码创建了一个数据框 temp；第 2 行代码使用 dbWriteTable 函数将 temp

上传到数据库，在数据库上创建了一个新表 TESTJDBC，它的第 1 个参数就是上一步创建的数据库连接 conn，第 2 个参数是在数据库上创建的表名称，第 3 个参数就是准备上传的数据框 temp；第 3 行代码同样在数据库上创建一个新表 TEMP，然后将 temp 写入进去；有趣的是 RJDBC 这个包是个半成品，它不能将数据添加到某个已经存在的表中，只能创建新表，也就是说不能更新表，所以如果需要更新，就只能用第 4 行代码的方式，使用 dbSendUpdate 函数对数据表进行更新，它的作用是在数据库上执行一条 SQL 语句，既然它能做到这一点，我们直接写一条 SQL 语句将一个表的数据添加到目标表就可以了，这样只需要将数据上传到临时表（2 句），再用 SQL 语句添加到目标表就可以了，“insert into TESTJDBC select \* from TEMP”这句 SQL 语句的意思分两部分：“select \* from TEMP”表示取出 TEMP 这个表中的所有数据，“insert into TESTJDBC”表示将刚刚取出的数据插入 TESTJDBC，这样通过第 3 行和第 4 行代码我们就完成了表的更新；第 5 行用于删除表 TEMP；第 6 行代码的 dbGetQuery 函数用于查询数据，第 1 个参数为数据库连接，第 2 个参数是一条 SQL 查询语句，除了直接将表删除以外，还有清空表的操作；第 7 行代码就是执行一条清空表 TESTJDBC 语句。

RJDBC 连接到 Hive 需要 R 所在的计算机也要在 Hive 集群内，而且已经安装了 Hive，然后才能通过 RJDBC 连接，上面这些活必须找专业的人去做，不要一个人偷偷地把集群搞瘫痪了，那就悲催了。

#### • RJDBC 连接 Hive

```
1 library(RJDBC)
2 jdbcDriver <- JDBC("org.apache.hive.jdbc.HiveDriver", list.files("/usr
/lib",pattern="jar$", full.names=T, recursive=TRUE))
3 conn <- dbConnect(jdbcDriver, sprintf('jdbc:hive2://172.35.28.75:
10000/default'), 'hive', 'hive_2016')
4 temp <- dbGetQuery(conn,"select * from test")
```

上述代码中，包加载完成后，可以去 Hive 官网下载 apache-hive-1.2.1-bin.tar.gz，然后解压在“/usr/lib”路径下，这个路径可以自己更改（见第 2 行代码）；第 3 行代码创建 Hive 连接，连接的 ip、端口等参数和普通数据库一样；第 4 行代码直接使用 SQL 语句在 Hive 里面查询数据。

以上我们连接了各种数据库，但是这些知识放在基础章节比较难以理解，大家可以跳过此节继续往下学习，等自己的知识体系构建起来之后再回头阅读此节。

## 1.5.3 乱码就像马赛克一样让人讨厌

字符串是一种比较特殊的类型，因为计算机只能处理数字，不能处理字符，所以字符要进行编码，就涉及一个字符串编码的问题。世界上不同的语言有千百种，每一种语言本质上都是一种符号系统，所以需要对它们进行编码，最早的 ASCII 编码只适应中文，所以中国提出了 GB2312（或称之为 ANSI），很多中文 Windows 系统中的软件就是用的这种编码，比如 R 和 Windows 的办公软件，但是如果每个国家总结一种字符编码，则难免会出现各种冲突，于是 Unicode 应时而生，Unicode 把所有语言符号统一在一套字符编码里，就不会出现因编码重复而出现的乱码了。

比如 Python 默认的是 Unicode 编码，但是 Unicode 一般使用 2 个以上字节编码，这样很占存储空间。于是，又出现了 UTF-8 编码，将 Unicode 编码转化为存储可变的，英文字符就用 1 个字节，

汉字通常为 3 个字节，只有特别生僻的字符才占 4~6 个字节，这里建议大家所有的数据最好统一为 UTF-8 编码。

Windows 下通过使用记事本打开文件，右击鼠标“另存为”时可以查看到文件的编码或者重新设置编码方式，如果将编码 UTF-8 的文件读入 Windows 系统下的 R 环境就会出现如下乱码的麻烦。

- 乱码：UTF-8

```
1 carhierarchy <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 1 章万事不  
只开头难/data/carhierarchyutf.csv", header = F, sep = ",", stringsAsFactors = F)  
2 head(carhierarchy)
```

上述代码中，可以看到当 Windows 系统读入 UTF-8 的中文时发生乱码甚至报错无法读入，比较笨的方法就是使用记事本打开文件，然后将文件另存为 ANSI 编码，再次读入即可，但是这种方法不太实用，在 Python 里面可以查看文件的编码，R 里面也有相关的包，只是效果不是很好。

- 检测文件编码

```
1 library(readr)  
2 guess_encoding("E:/业余/zimeiti/探寻数据背后的逻辑：R 语言数据挖掘之道  
/bookwriting/第 1 章万事不只开头难/data/carhierarchyutf.csv", n_max = 10)  
# encoding confidence  
# 1 UTF-8 1
```

上述代码中 readr 包中的 guess\_encoding 可以探测一个文件的编码，只需要指定文件的路径即可，n\_max 参数用于设置检测前多少行，它会返回编码的类型、可信度。可信度越高，说明数据使用某种编码的可能性越大，这也告诉我们一个文件可能存在多种编码，甚至一段文字就存在多种编码，非常麻烦，所以编码最好在源头就将其统一。

- 更改数据编码

```
1 carhierarchy$V1 <- iconv(carhierarchy$V1, from = "utf-8", to = "gb2312")  
2 carhierarchy
```

上述代码中使用 iconv 更改数据编码，iconv 函数可以将数据从一种编码转化为另外一种编码，第 1 个参数为数据，参数 from 指定数据现在的编码类型，参数 to 指定将要转化的数据类型，这样就将数据的编码类型转化了。另外在读取数据时我们也可以设定数据的编码类型，如下所示。

- 设定数据编码

```
1 carhierarchy <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 1 章万事不  
只开头难/data/carhierarchyutf.csv", header = F, sep = ",", stringsAsFactors = F,  
fileEncoding = "utf-8")  
2 head(carhierarchy)
```

在读取数据时，可以使用 fileEncoding 设定编码，但是有时候不一定有用，因为同一个数据集可能有多种编码，另外 R 的不同版本对这个支持也不是很好。

这一节是第 1 章的最后一节，到这里大家应该对 R 语言有一个基础的认识了，第 1 章的内容很多，但是它们又极其重要，因为以后的数据挖掘大厦都要用这些基础的砖块构建，对第 1 章的内容理解越透彻，以后的应用越得心应手。



## 第 2 章

# 数据探索，招招都是利器

第 1 章我们讲述了 R 语言的基础知识，了解第 1 章的内容只能说我们捡到了一块陨铁，要将其炼成鱼肠还需要功夫。这一章我们主要用于摆脱其他软件，特别是 Excel 的思维定势，让自己成为一个真正能够使用 R 语言行走于日常工作的小能手。任何一份数据或者数据分析的项目都要先对数据进行必须的探索，探索的目的不是为了找出数据的毛病或者是例行公事，而是增进自己对数据的了解和理解。说到这里，可能有些读者要笑我故弄玄虚，因为大多数人认为数据探索可有可无，只要模型好就可以了。那有一天老板问你：数据有哪些字段？各个字段的含义是什么？它们有没有计算转化关系？有多少字符字段？有多少数值字段？各个字段有没有缺失，缺失比例是多少？各个字段最大值是多少最小值是多少？有没有异常值？哪些是预估的数据，哪些是实际值？字符字段有没有因子水平？各个水平的含义是什么？有没有定序型的因子，需不需要转化？等等诸如此类的问题，你怎么办？更重要的是，这些前期的了解能够避免出现项目做了一半才发现重大的数据质量问题的情况。

通过本章我们争取夯实自己数据清洗、数据处理的基础，我刚开始学习 R 时就去尝试各种模型，直到工作后才发现，自己的水平用于数据处理简直捉襟见肘，衡量自己 R 语言水平的一个最直接的标准就是你最近还想得起用 Excel 处理数据吗？所以本章的主要目的在于以 R 语言的方法轻松解决日常工作中数据处理的问题。

### 2.1 不要在工作后才认识“脏数据”

课堂上需要你快速读入应用案例的数据集，而实际工作中需要你将  $n$  份调研问卷或者说  $n$  份格式混乱的数据集整理为干净的数据，就像应用案例的数据集一样整洁，这就是课堂和工作的区别。想象中数据应该长成这个样子，如图 2-1 所示。

省份	地级城市	单位名称	医院级别	医院等次	床位数	诊疗人次	总收入
北京	北京	北京荣军医院	未评级	未评等	20	4420	6540
北京	北京	北京航天器制造公司北京东城航天医院	未评级	未评等	20	40000	7324
北京	北京	北京市隆福医院	二级	甲等	251	279961	200836
北京	北京	首都医科大学中医药学院附属鼓楼中医医院	二级	甲等	181	112322	112050
北京	北京	北京市东四中医医院	一级	甲等	60	23106	26962
北京	北京	北京普康中医医院	未评级	未评等	20	14129	3062
北京	北京	北京金典糖尿病医院	未评级	未评等	20	2194	5086
北京	北京	北京同力医院	未评级	未评等	20	3987	1022

图 2-1

而实际上它们经过疯狂生长，往往到你手里时是这个样子，如图 2-2 所示。

```
<a href="https://movie.douban.com/subject/25986180/" class="">
  釜山行
</a>
<span style="font-size:12px;">尸速列车(台) / 尸杀列车(港)</span>
</a>
<p class="pl">2016-07-20(韩国) / 孔侑 / 郑有美 / 马东锡 / 金秀安 / 金义城 / 崔宇植 / 安昭熙 / 沈恩京 / 韩国 / 延尚昊 / 118
分钟 / 釜山行 / 动作 / 惊悚 / 灾难 / 延尚昊 Sang-ho Yeon / 韩语</p>

<div class="star clearfix">
<span class="allstar45"></span>
<span class="rating_nums">8.3</span>
<span class="pl">(130729人评价)</span>
```

图 2-2

提前认识脏数据，能让你的工作更加轻松。在现实的工作场景中处理脏数据的能力和效率与对模型、业务的理解同样重要。

直面脏数据时的沉着是数据挖掘工程师成熟的表现。

所以数据拿到之后的第一步是看看数据有多脏，清洗整理数据，在数据整理出来之前不要去幻想任何模型。如果数据没有清洗干净，则说明你对数据的情况了解甚少，在这种情况下向别人保证模型的效果都是不负责任的，也就是说在数据清洗或者看清庐山真面目之前说话要留有足够的余地。

数据清洗一般包括数据对象化、质量清洗和数据抽样，在前言中已经讲过，这里主要指数据的质量清洗，质量清洗一般要做缺失值处理、异常值预警、字符数据清洗三个主要工作。

### 2.1.1 以老板信服的方式处理缺失数据

缺失值是数据处理过程中经常碰到的一个现象，虽然我们将要聊到很多看似高大上的缺失值处理方法，但我个人建议，如果数据允许，还是尽量把含有缺失值的观测样本删除掉，不要自欺欺人。比较苦闷的是，很多情况下删除了缺失值之后数据也所剩无几了，这种情况下为了照顾老板的脸色，我们还是要想一想办法，不然显得有点不合时宜。

#### 数据准备

不妨使用 mlbench 包中的 BostonHousing 数据集讨论几种缺失值处理的方法，但是 BostonHousing 数据集并不存在缺失值，所以需要随机制造一些缺失值。

- 调入数据

```
1 install.packages("mlbench")
2 data(BostonHousing, package = "mlbench")
3 original <- BostonHousing
4 set.seed(1603)
5 BostonHousing[sample(1:nrow(BostonHousing), 40), "rad"] <- NA
6 set.seed(1603)
7 BostonHousing[sample(1:nrow(BostonHousing), 40), "ptratio"] <- NA
```

上述代码中，第1行代码安装了 mlbench 包，因为要使用其中的数据 BostonHousing；第2行代码使用 data 函数加载 BostonHousing 数据，因为我们要对数据集 BostonHousing 进行少许更改；第3行代码将 BostonHousing 赋值给一个新对象；第4行代码的 set.seed 函数用于设置随机种子，因为下面要使用到随机抽样的函数 sample，为了保证你随机抽样的结果和我的一样，所以我们设置了一个密钥 1603，就是随机种子，保证在不同的计算机上生成的随机数字是一样的；第5行代码使用 nrow 函数获取数据集 BostonHousing 有多少行，1:nrow(BostonHousing) 产生一个公差为 1 的等差序列，可以单独在终端执行下看看结果，sample 用于从刚刚生成的序列中进行随机抽样，抽出 40 个数字，然后使用这些数字作为行编号在 BostonHousing 中筛选，将筛选出来的数据中相应的 rad 列的值替换为 NA，即缺失值；第6行和第7行代码使用相同的方法筛选出 40 行数据并将其中的 ptratio 替换为 NA 值，这样具有缺失值的数据集就制造好了。

在进行缺失值处理之前，首先应该大概看一下数据的缺失情况，比如重点是哪些维度的数据缺失等，可以使用 mice 包中的 md.pattern 函数查看缺失值存在的模式，如下所示。

- 快速统计缺失值

```
1 library(mice)
2 md.pattern(BostonHousing)
#      crim zn indus chas nox rm age dis tax b lstat medv rad ptratio
# 431    1  1    1    1  1  1  1  1  1  1  1  1    1  0
#  35    1  1    1    1  1  1  1  1  1  1  1  0    1  1
#  35    1  1    1    1  1  1  1  1  1  1  1  1    0  1
#   5    1  1    1    1  1  1  1  1  1  1  0    0  2
#      0  0    0    0  0  0  0  0  0  0  0  0  40  40  80
```

上述代码中，不存在缺失值的样本为 431 个，仅 rad 变量缺失的观测样本为 35 个，仅 ptratio 缺失的观测样本为 35 个，rad 和 ptratio 同时缺失的共有 5 个观测样本；整个数据集共缺失 80 个数据点，涉及 75 个观测样本，其中 rad 和 ptratio 各 40 个，其他变量无缺失数据点。

那么如何处理这些缺失值？至少有以下 4 种方法。

#### 方法 1：删除样本

如果数据量非常大，删除一些缺失样本不会影响训练集，我还是强烈建议直接把缺失值删除，或者在建模时设置参数忽略缺失值（na.action = na.omit），但是在删除之前还是要做一下分析的。

#### 删除缺失数据守则

（1）删除后必须保证有足够的训练样本用于建模，保证模型的效能。

(2) 删除后数据的分布结构不发生变化, 比如删除后有些分类比例明显减少等, 但分布不仅仅指分类。

- 建模删除缺失值

```
lm(formula = medv ~ ptratio + rad, data = BostonHousing, na.action = na.omit)
```

lm 就是线性模型 (Linear Models) 的缩写, 用于构建简单的线性模型, 比如上述代码用 ptratio、rad 作为自变量预测因变量 medv, 首先要设置方程式, 即简单表达它们之间的预测与被预测关系 (medv ~ ptratio + rad), 然后使用 data 参数指定数据集, na.action 参数是我们的重点, 这里指定是否移除包含缺失值的数据, na.omit 表示忽视这类数据, 也就是将缺失的数据移除之后再构建预测模型, 详细实例参看第 4 章。

方法 2: 删除变量

删除变量则是从另外一个角度处理问题, 如果一个变量含有比其他变量都要多的缺失值, 移除它可以保存大部分观测样本, 建议还是将这类变量移除, 除非它含有重大的业务意义, 这里就存在一个评价变量重要性和损失样本完整性的平衡问题。

方法 3: 均值、中位数、众数填充

均值、中位数、众数填充是一种十分武断简单粗暴的方法, 但如果变量的方差很小, 而且对因变量影响较小时, 也是能够接受的, 因为它们容易解释、方便操作。

- 均值、中位数、众数填充

```
1 install.packages("Hmisc")
2 library(Hmisc)
3 impute(BostonHousing$ptratio, mean) # 均值填充
# BostonHousing$ptratio <- as.vector(impute(BostonHousing$ptratio, median))
4 impute(BostonHousing$ptratio, median) # 中位数填充
5 impute(BostonHousing$ptratio, 20) # 指定数据填充
```

上述代码中, 第 3 行代码的 Hmisc 包的 impute 函数专门用于各种方法的填充缺失值, ptratio 均值填充 ptratio 的缺失值, 其中带星号的为填充值, 这里为了演示其他方法没有将结果填充到数据集里面, 不然缺失值就没了, 如果你想直接填充可以参考注释的那句, 只需要将 impute 的结果转化为 vector, 然后赋值给原数据集中的 ptratio 列即可; 第 4 行代码使用中位数填充; 第 5 行代码使用固定数值填充, 这里使用 20 填充所有值。

如果你要手动完成也可以, 如下所示。

```
# BostonHousing$ptratio[is.na(BostonHousing$ptratio)] <- mean(BostonHousing$ptratio, na.rm = T)
```

操作时不要真的执行上面的代码, 除非创建了临时数据集, 不然缺失值被填充了, 需要时又要重新制造缺失值。is.na(BostonHousing\$ptratio) 生成一个逻辑序列, 用这个逻辑序列提取 BostonHousing\$ptratio 中的缺失值, 然后用 ptratio 列的均值替换这些缺失值, 这里在计算均值时已经使用 na.rm 参数将缺失值排除在外了。

因为保存了原来的真实数据 (original), 所以我们可以测试一下均值填充策略的效果, 评价指标 mae、mse、rmse、mape 等, 这些评价指标在第 10 章有详述, 如下所示。

- 测试效果

```
1 install.packages("DMwR")
2 library(DMwR)
3 actuals <- original$ptratio[is.na(BostonHousing$ptratio)]
4 predicted <- rep(mean(BostonHousing$ptratio, na.rm=T), length(actuals))
5 regr.eval(actuals, predicted)
#>      mae      mse      rmse      mape
#> 1.62324034 4.19306071 2.04769644 0.09545664
```

上述代码中，DMwR 包的 `regr.eval` 函数用于评价真实值和预测值之间的差异，计算了 `mae`、`mse`、`rmse`、`mape` 等常见的评价指标。第 3 行代码是从真实数据（original）中 `ptratio` 被替换为 NA 值的那些数据的真实值，筛选是用 `is.na(BostonHousing$ptratio)` 产生的逻辑值，因为 `BostonHousing$ptratio` 对应的缺失值刚好对应到原数据的真实值；第 4 行代码使用均值预测缺失值的大小，直接使用均值函数计算就可以了，记得计算时要移除缺失值，然后 `actuals` 中有几个值就将均值使用 `rep` 函数复制几次即可，将复制好的记过赋值给新对象 `predicted`；第 5 行代码中的 `regr.eval` 函数的第 1 个参数指定真实值，第 2 个参数指定预测值，即可以输出评价指标了。

如果 `regr.eval` 输出的是 NaN 值，则表示你已经将原来的数据替换掉了，不要执行测试代码，如果执行了测试代码就要重新来。

#### 方法 4：模型预测缺失值

模型预测填充缺失值是比较高端的方法，但是高端的方法不一定高效，因为模型算法的种类多样，预测的方法也比较多，这里主要介绍 KNN、决策树和 `mice` 方法，如果对这些模型算法不是很了解，可以通过百度搜索或者参看第 7 章。

任何时候都不要轻易把简单复杂化。

#### (1) KNN 预测填充

DMwR 包的 `knnImputation` 函数可以使用 KNN 聚类完成缺失值填充，理论上可以简单理解为通过欧式距离算出  $k$  个最邻近的观测样本，然后计算加权（根据距离加权）平均值，最后把均值作为预测值填充。这个方法可以将整个数据框传给函数，函数一次填充所有的缺失值，缺点是速度比较慢，需要提醒读者的是，千万不要将因变量 `medv` 也一起输入，因为数据会利用因变量提供的信息进行预测，所谓因变量就是你将要建模预测的变量，这里假想为 `medv`。

- KNN 预测填充

```
1 library(DMwR)
2 knnOutput <- knnImputation(BostonHousing[, !names(BostonHousing) %in% "medv"])
3 anyNA(knnOutput)
# [1] FALSE
```

上述代码中，将数据中除了 `medv` 列以外的所有数据维度丢给 `knnImputation` 函数，它会构建模型完成对 `medv` 列中缺失值的填充，`!names(BostonHousing) %in% "medv"` 表示剔除名称为 `medv` 的列，`%in%` 函数前面已经介绍过了；`anyNA` 函数用于查看数据框中是否包含缺失值。

需要提醒读者的是，这里面涉及模型和算法的知识我们将会在第 7 章中详细介绍，当然读者也要补充一些统计学知识。

- 测试效果

```
1 actuals <- original$ptratio[is.na(BostonHousing$ptratio)]
2 predicted <- knnOutput[is.na(BostonHousing$ptratio), "ptratio"]
3 regr.eval(actuals, predicted)
#>      mae      mse      rmse      mape
#> 1.00188715 1.97910183 1.40680554 0.05859526
```

与均值填充策略比较，平均绝对百分比误差 (mape) 提升了 39%。

### (2) 决策树填充

除了速度以外，knnImputation 函数对于因子变量填充也比较无能，rpart 和 mice 在处理数值变量和分类变量方面都比较灵活，对于 rpart，你只需要输入一个不包含空值的变量就可以了，下面使用 rpart 完成缺失值填充。

- 决策树填充缺失值

```
1 library(rpart)
2 class_mod <- rpart(rad ~ . - medv, data=BostonHousing[!is.na(
  BostonHousing$rad), ], method="class", na.action= na.omit) # rad 是因子变量
3 anova_mod <- rpart(ptratio ~ . - medv, data = BostonHousing[!is.na(
  BostonHousing$ptratio), ], method="anova", na.action=na.omit) # ptratio 是数
  # 值变量
4 rad_pred <- predict(class_mod, BostonHousing[is.na(BostonHousing$rad), ])
5 ptratio_pred <- predict(anova_mod, BostonHousing[is.na(BostonHousing$ ptratio),])
```

上述代码中，第 1 行代码加载 rpart 包；第 2 行代码使用 rpart 函数构建一个决策树模型，它的函数式写得比较简单， $rad \sim . - medv$  表示将 rad 作为目标变量，也就是自变量，使用其他变量（除了 medv 以外的变量，减号表示剔除）作为因变量来预测 rad，rad 是因子变量，填充因子变量，可以将 rpart 函数的 method 参数设置为“class”，即分类任务；对于数值变量如 ptratio，可以将 method 设置为 anova，即方差分析，同样需要将因变量 medv 移除，如第 3 行代码；通过第 2 行和第 3 行代码我们分别建立了两个模型，分别存储为 class\_mod 和 anova\_mod，用于预测 rad 和 ptratio；predict 函数使用已经构建好的模型和模型的自变量预测目标变量，比如第 4 行代码我们使用分类模型 class\_mod 预测缺失的 rad，第 5 行代码使用 anova\_mod 预测数值型变量 ptratio，predict 函数的第 1 个参数为已经构建的模型，第 2 个参数为输入模型的数据，这里数据可以是新数据、测试数据等，但是一定要保证包含所有用于构建模型的自变量。

什么是自变量和因变量？最好去看统计学知识，实在不行，如  $rad \sim . - medv$  公式中符号“~”之前的是因变量也叫目标变量是被预测的变量，之后就是自变量，用于提供预测目标变量的信息。

- 测试效果

```
1 actuals <- original$ptratio[is.na(BostonHousing$ptratio)]
2 predicted <- ptratio_pred
3 regr.eval(actuals, predicted)
```

```
#>      mae      mse      rmse      mape
#> 0.71061673 0.99693845 0.99846805 0.04099908
```

MAPE 指标相对 KNN 方法，其测试效果又提高了不少。

- 测试效果

```
1 actuals <- original$rad[is.na(BostonHousing$rad)]
2 predicted <- as.numeric(colnames(rad_pred)[apply(rad_pred, 1, which.
max)])
3 mean(actuals != predicted)
#> 0.25
```

分类变量的准确率达到了 75%，效果差强人意。

### (3) MICE 填充

基于链式方程的多重插补方法，也称为 MICE（Multiple Imputation by Chained Equations）。MICE 方法与其他方法略有不同，基本上可以分两步理解，mice 函数使用蒙特卡罗随机选择数据创建模型，complete 函数基于模型返回完整的数据集，mice 函数会预测多个完整的数据集，每一个数据集填充的缺失值都不一样，complete 函数返回这些数据集中的某一个或多个，默认情况下返回第一个数据集，其使用的算法可以通过 method 参数选择，这里选择了随机森林（RF）算法。

- MICE 预测填充

```
1 library(mice)
2 miceMod <- mice(BostonHousing[, !names(BostonHousing) %in% "medv"],
method= "rf") # 基于随机森林的填充
3 miceOutput <- complete(miceMod)
4 anyNA(miceOutput)
#> FALSE
```

上述代码中，第 1 行代码加载 mice 包，没有安装的读者先自行安装；第 2 行代码使用 mice 函数构建模型并完成缺失值填充，这里仍然剔除了 medv，把剩余的其他变量作为数据输入模型，method 指定随机森林模型，这一句函数里包含了模型构建、预测、填充等动作；第 3 行代码输出填充完整的数据集；第 4 行代码使用 anyNA 函数查看是否还存在缺失值。

下面测试一下填充效果，不难发现对于数值型变量 ptratio 来说，基于随机森林的 MICE 是最优秀的方式，MAPE 又有大幅提升。

- 测试效果

```
1 actuals <- original$ptratio[is.na(BostonHousing$ptratio)]
2 predicted <- miceOutput[is.na(BostonHousing$ptratio), "ptratio"]
3 regr.eval(actuals, predicted)
#>      mae      mse      rmse      mape
#> 0.36500000 0.78100000 0.88374204 0.02121326
```

对于分类型因子变量的预测效果准确率也提高到了 85%，很不错。

- 测试分类效果

```
1 actuals <- original$rad[is.na(BostonHousing$rad)]
2 predicted <- miceOutput[is.na(BostonHousing$rad), "rad"]
```

```
3 sum(actuals != predicted)/length(actuals)
#> 0.15
```

对于分类变量我们是不能使用上述评价指标评比的，只能使用准确率，分类能预测模型是分对了还是分错了。上述代码中，前两行代码筛出真实的 rad 值和预测的 rad 分类值；第 3 行代码计算分类错误的比率，`actuals != predicted` 判断两者是否不相等，不相等返回真，否则返回假，最终生成一个逻辑序列，然后 `sum` 函数对这个逻辑序列中的真值求和，也就是计算出真实值和预测值不相等的个数，再除以总数，就得出了分类错误的比例。

尽管上面我们给出了处理缺失值的几个方案，但是并不足以证明在现实情况中孰优孰劣，上述方法在你处理缺失值时绝对值得测试一下。

其实很多时候我们在争论哪个方案优越时，无论选择哪个方案都不会带来太大的优势，在这种状态下与其无休止地争论，倒不如妥协，尽快达成共识，抓紧时间将方案实现。

另外，我不得不告诉你一个让人失望透顶的现实，在你实施上述方法之前，请先与业务人员确认，让他们想办法把缺失的数据找出来，如果他们没有办法，才能采用以上备选方案。

### 2.1.2 异常值预警

所谓异常值就是那些超出平常方位的数据，又可以称为奇异值、离群值等，这类数据介乎事实和错误之间，前者表示它们可能事实上就应该是这个样子，只是比其他数据更高（更低）而已，比如具有季节性的数据或者某次爆炸消息引起的特殊流量等，后者表示它们可能就是录入错误，这类错误比缺失更加隐蔽。

但从我们数据挖掘者的角度来说，这类数据不能轻易自行处理，只能找出来和业务人员一起分析，查找背后的原因。而异常值从统计学上定义如下，当然这也是我们在项目中经常用到的方法。

#### 异常值

$x > \text{上百分位数} + 1.5 \times (\text{上百分位数} - \text{下百分位数})$

$x < \text{下百分位数} - 1.5 \times (\text{上百分位数} - \text{下百分位数})$

如果数值大于数据上百分位数加  $1.5 \times (\text{上百分位数} - \text{下百分位数})$ ，或者小于下百分位数减去  $1.5 \times (\text{上百分位数} - \text{下百分位数})$ ，我们就认为是轻度异常，这个规则可能有些宽松，我们可以使用下面的规则定义出极端异常值。

#### 极端异常值

$x > \text{上百分位数} + 3 \times (\text{上百分位数} - \text{下百分位数})$

$x < \text{下百分位数} - 3 \times (\text{上百分位数} - \text{下百分位数})$

如果对百分位数不是很了解可以自行参阅其他资料，这实在是十分基础的统计学内容，很多读者觉得 R 语言很难学，其实主要难在统计学。但是我建议读者还是要自备一本统计学的书，因为很多东西都是建立在基础的统计学之上的，包括你向往的高级算法和模型。下面我们就分析下全国医院收入中的异常值。



- 载入数据

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第2章数据探索招
招都是利器/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)
2 tail(hospital)
3 str(hospital)
4 q <- quantile(x = hospital$总收入, probs = c(.25, .75))
5 q
6 outer.low <- q[1] - 1.5*(q[2]-q[1])
7 outer.low
8 outer.upper <- q[2] + 1.5*(q[2]-q[1])
9 outer.upper
10 temp <- hospital[hospital$总收入 < outer.low|hospital$总收入 > outer.upper, ]
```

上述代码中，第1行代码读取医院的数据；第2行代码查看数据的最后6行；第3行代码查看数据的结构，数据包含5个字符变量和3个数值型变量；第4行代码的 quantile 函数用于计算数据的百分位数，第1个参数 x 用于指定数据，第2个参数用于指定百分位点，这里计算下百分位数和上百分位数；第5行代码查看计算结果；第6行代码计算异常值的下边线，取下百分位数减去1.5倍的上下百分位数之间的差值；第7行查看下边线计算结果；第8行和第9行代码计算上边线并查看结果；第10行代码筛选出总收入低于下边线或者高于上边线的医院数据，前面讲过这类如“hospital\$总收入 < outer.low”比较返回的是一个和“hospital\$总收入”等长的逻辑序列，正好用于筛选提取，“|”在 R 里面表示或的关系，“&”表示且的关系，这些符号类型应该是各种语言都通用的。

除了上述阈值以外，还可以通过可视化的方法进行分析，比如箱线图，使用数据的最小值、第一四分位数（下百分位数）、中位数、第三四分位数（上百分位数）、最大值勾勒一个类似箱体的图，反映数据分布的中心位置和分布情况，可以粗略地看出数据是否具有对称性，如下所示。

- 箱线图查看离群值

```
1 boxplot(hospital$总收入)
2 hist(hospital$总收入, breaks = 1000)
```

从箱线图上分析，绝大多数医院的收入非常低，它们一般是级别较低的医院或社区诊所，全国医院收入可以看成明显的右偏态分布，boxplot 函数用于绘制初级的箱线图，hist 用于绘制数据的频率图，breaks 参数用于设置频率统计的组距，这些图将在第3章中讲解更加专业的绘制方法。

异常值的检测方法上面已经讲了，但脱离了概念，你可能已经发现这只是一个阈值划分的问题，怎么设定阈值不仅只有上面讲解的方法，还可以用均值和方差等方法构筑，完全根据数据和业务需要而定。另外，异常值需要两个层次理解，其一异常值是错误还是异常？其二异常背后的原因是什么？找到这类原因可能就是业务突破的利器或者至少是市场营销中的关键点。

## 2.1.3 字符处理正则表达式不再是天书

字符处理应该是数据清洗中最为常见的操作，而且折磨你意志、浪费你时间、考验你耐心的恰恰是字符串的处理，它没有固定的模式，因项目而异，比如在一个项目中可能需要移除所有的英文

字符，另外一个项目又要求你提取保留具有特殊含义的英文缩写，所以这种工作无定式。

处理这种工作只有一个要诀：善于将 R 语言的函数和正则表达式结合。需要注意的是通过一定的方式提高速度。我们前面已经讲了很多字符串相关的函数，至于具体的实例将在后面的章节中用到时再仔细分析讲解。

## 2.2 数据透视、数据整形、关联融合与批量处理

数据透视与透视表一样，主要做一些数据汇总的工作，一般用于模型构建之前的指标构建，数据整形主要是数据结构的变换，为了整理出用于模型输入的数据结构，这些都是数据分析工作中的日常，也是所谓的基本功。

### 2.2.1 还忘不掉 Excel 的数据透视表吗

Excel 的透视表相信大家都用过，就是通过傻瓜式的拖曳各种维度对数据进行汇总、计数等常见计算的非常灵活的报表制作方式，其本质是对所选维度进行一些汇总、分组计算。R 自然也可以制作各种报表，但是不能像 Excel 那样傻瓜式操作，它有其自身的优势，比如应用到的计算公式你可以通过自编函数实现，这就比计算方式比较固定的 Excel 优势明显。另外，比如制作  $n$  张报表，Excel 可能需要点击  $n$  次，而 R 语言脚本写好后就可以批量产出。我们这一节主要学习制作透视表时常见的汇总类操作。

计数就是数一数有多少个，比如我要统计全国不同等级医院的个数，看一看全国不同省份不同等级医院的个数，可以称得上是最基础的报表统计方式，代码如下所示。

- 简单计数 (table)

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招  
招都是利器/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)  
2 table(hospital$医院级别)  
3 temp <- table(hospital$省份,hospital$医院级别)  
4 head(temp)  
5 temp <- data.frame(temp)  
6 names(temp) <- c("province", "class", "freq")  
7 head(temp)
```

上述代码中，第 1 行代码读取数据，这个数据集包含了全国近两万家医院的基础数据；第 2 行代码对全国不同级别的医院计数，这里使用 table 函数，将医院的级别列作为参数即可，它会计算每个级别出现的个数（首先默认每一行数据代表一家医院，医院不能有重复）；那如果我们要统计不同省份不同级别医院的个数怎么做？第 3 行代码仍然使用 table 函数，只不过需要将省份作为第 1 个参数，医院级别作为第 2 个参数就可以了，结果是一个 table 对象，每一列是一种医院级别的统计数字，

需要转化为 data.frame；第 5 行代码将结果 temp 转化为数据框，数据变成了 3 列：省份、级别和数量；第 6 行代码对数据框的列名重新命名；第 7 行代码查看最终结果。以上就是简单计数的透视表方法，当然 R 里面还有很多能够实现的函数，但是 table 是一个基础函数，不能避而不谈，另外，当数据量比较大时，基础函数的速度非常快，提醒大家尽量不要使用中文命名任何对象。

除了简单计数以外，我们可能还会进行其他比较复杂的计算，如求每一个省份不同等级医院收入的总和、均值、中位数等，或者其他更加复杂的计算，虽然使用一些基础函数经过比较多的步骤也能实现，但在 R 里面可能一行代码就搞定了，如下所示。

- 高级透视 (1)

```
1 saletotal <- aggregate(总收入 ~ 省份, data = hospital, FUN = sum)
2 pcsale <- aggregate(总收入 ~ 省份 + 医院级别, data = hospital, FUN = sum)
3 pcsalemean <- aggregate(总收入 ~ 省份 + 医院级别, data = hospital, FUN = mean)
4 pcsalemedian <- aggregate(总收入 ~ 省份 + 医院级别, data = hospital, FUN =
median)
5 pcsalemedian <- aggregate(cbind(总收入, 诊疗人次) ~ 省份 + 医院级别, data =
hospital, FUN = mean)
```

上述代码中，第 1 行代码将总收入按省份分组求和，aggregate 函数的第 1 个参数是一个形如 formula(公式)的表达式，表示将波浪符号前面的变量按照波浪符号后面的变量进行分组，然后将分组的结果应用到后面的函数 FUN，FUN 参数调用指定的函数进行计算，这里调用了求和函数 sum，data 参数用于指定数据集，数据集必须包含表达式中用到的变量；第 2 行代码将总收入按照省份、医院级别分组后求和；第 3 行代码将总收入按照省份、医院级别分组后求均值，也就是求不同省份不同级别的医院平均收入；第 4 行代码将总收入按照省份、医院级别分组后求中位数；第 5 行代码将总收入、诊疗人次按照省份和医院级别分组然后求分别对总收入和诊疗人次求均值，分组之前先将总收入、诊疗人次使用 cbind 捆绑在一起。

一个 aggregate 函数基本上能做出所有 Excel 透视表的动作，除了它以外还应该接触 dplyr 包的一些函数，其中的 group\_by 函数和 summarise 函数常被用来完成 aggregate 的任务，速度比较快，如下所示。

- 高级透视 (2)

```
1 library(dplyr)
2 grouped <- group_by(hospital, 省份, 医院级别)
3 temp <- summarise(grouped, salemean = mean(总收入), salesd = sd(总收入))
4 temp <- data.frame(temp)
```

上述代码中，第 1 行代码加载包；第 2 行代码将数据集按照省份和医院级别分组，group\_by 函数用于分组，第 1 个参数指定分组的数据集，后面的参数指定分组的变量；第 3 行代码将分好组的数据对象丢给 summarise 函数，作为它的第 1 个参数及数据集，第 2 个参数表示在结果中新添加一列，名称为 salemean，调用 mean 函数对总收入求均值，然后将结果赋值给新列 salemean，第 3 个参数在结果中新添一列取名为 salesd，然后调用 sd 函数对总收入求方差，将结果赋值给新列 salesd；第 4 行代码将结果转化为数据框即可，这样 summarise 一次调用了两个函数，同时求得了不同省份

不同级别的医院总收入的均值和方差。

上一步产生了中间变量 `grouped`，在数据集很大的情况下，中间变量会吞噬掉大量的内存，这样可能会导致内存不足，另外给中间变量起名字也是件让人头疼的事情，而 `dplyr` 包的管道函数解决了这一问题，如下所示。

- 管道函数

```
1 temp <- hospital %>% group_by(省份, 医院级别) %>% summarise(salemean =  
mean(总收入), salesd = sd(总收入))
```

`dplyr` 包里面的管道函数 (`%>%`) 很有用，它可以理解为像一条管道一样将数据流下来，比如上例中，我们将 `hospital` 通过管道函数 `%>%` 专递给了分组函数 `group_by`，分组函数将它按照省份、医院级别分组后，没有产生中间变量，而是直接转递给了 `summarise` 函数进行均值方差的计算，然后将计算结果赋值给了 `temp`，与第一次使用减少了中间变量 `grouped`，使代码更加简洁。

但是需要提醒大家的是，在使用了 `dplyr` 包的脚本中，就不要再加载 `plyr` 了，因为它们两个经常冲突，`dplyr` 就是 `plyr` 的升级版，它们的函数名大部分一样，重复加载会造成新的覆盖旧的。

### 2.2.2 你能给数据做整形手术吗：long 型和 wide 型

所谓数据整形，说简单点就是改变数据的摆放形式，下面我们要讲一个很重要的数据整形方式，它几乎和透视表一样重要，即 long 型和 wide 型数据表变换。之所以这么强调其重要性，主要是因为数据分析中经常用到，另外 R 的基础作图和 `ggplot2` 作图所需要的数据是不一样的，对于 `base R` 来说，wide 型数据更适合，而对于 `ggplot2`，则 long 型比较方便，为什么是这样？你可以私信 Hadley Wickham 了解。

我们首先认识一下 long 型和 wide 型数据的区别，wide 型表如下。

- wide 型表

```
1 datawide <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招  
招都是利器/datawide.csv", header = T, sep = ",", stringsAsFactors = F)  
2 head(datawide)
```

上面产生的数据框即可称之为 wide 型数据，第 1 列为省份，其实它的后 4 列可以合并成一列作为医院等级，然后再将数值合并成一列作为总收入就变成了 long 型数据，合并后结果如下。

- long 型表

```
1 datalong <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招  
招都是利器/data/datalong.csv", header = T, sep = ",", stringsAsFactors = F)  
2 head(datalong)
```

long 型把 wide 型的最后 4 列的列名组成了一个新的列，作为医院级别，然后相应的数值组成了另一列，作为总收入，数据变长了。这就是 long 型与 wide 型的区别，这种数据之间的变换很常见，是数据整形的必备技能之一。比较常用的转换包为 `reshape2`，这个包里的 `melt` 函数和 `dcast` 函数分别对应以上任务。下面以上面两个数据框为例进行 wide 型与 long 型之间的转化。

- 将 wide 型转化为 long 型

```

1 library(reshape2)
2 temp <- melt(data = datawide, id.vars = c("省份"), measure.vars = c("二
级", "三级", "未评级", "一级"), variable.name = "医院级别", value.name = "总收入")
3 head(temp)
4 temp <- melt(datawide, id.vars = c("省份"), variable.name = "医院级别",
value.name = "总收入")

```

查看一下数据框 temp 和我们之前的 datalong 就发现一模一样了，reshape2 包中的 melt 函数就是将 wide 型数据转化为 long 型数据，顾名思义，melt 就是融合的意思。第 1 个参数 data 为将要转化的数据框；参数 id.vars 用来指定 id 列，所谓 id 列就是保持原来的排列方式不改变的那些列，比如我们希望“省份”保持原来的位置，它可以是一个向量，需要保持更多列不变，在后面添加就是了；measure.vars 指定要融合的列，variable.name 用来指定融合后的分类变量列的名字，我们这里取名为医院级别；value.name 用来指融合后的数值列的名称，这里取名为总收入，这样一个函数就完成了从 wide 到 long 的整形；有时候我们除了保留 id 列不变以外，其他列统统融合，这样在 measure.vars 中一个一个地写出列名就比较麻烦，这种情况可以不指定 measure.vars，默认表示去除 id 列以外数据框的其他列都参与融合，如上面第 4 行代码。

- 将 long 型转化为 wide 型

```

1 temp <- dcast(data = datalong, formula = 省份 ~ 医院级别, value.var = "总
收入", fun.aggregate = sum)
2 head(temp)

```

可以看到，temp 就由 long 型转化为和 datawide 一模一样的 wide 型数据了，dcast 函数指的是 data.frame casting 的意思，即数据框转换。第 1 个参数 data 指定将要转化的数据框；第 2 个参数 formula 是一个表达式，前半部为 id 列，就是那些保持原来位置不变的列，如果需要添加多个 id 列就用加号隔开，如“省份 + 城市”，后半部是要差分的分类列，一般只有一列，前后用表达式分隔符“~”分隔开；value.var 参数指定要差分的数值列；fun.aggregate 指定数据透视的函数，即指定如果出现分类变量相同的数据，其 value 值怎么计算，这样就完成了从 long 型到 wide 型的延展。

另外，在进行 long 型转 wide 型时，可能会只选择一到两个 id 列，而不是选择所有的 id 列保持不变，数据有可能出现重复行，比如下面的数据：

```

1 dcastfun <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招
招都是利器/data/dcastfun.csv", header = T, sep = ",", stringsAsFactors = F)
2 head(dcastfun)

```

如果只选择省份这一列作为 id 列，“北京+二级”就会出现两次，那 dcast 函数就要知道怎么计算这两条数据，是进行加减乘除等哪一种计算？很多读者写代码时都会省略 fun.aggregate 参数的指定，但是这时你一定要保证数据没有出现上述重复现象，我们看一下它们的区别，如下所示。

- 是否指定 function 参数的区别

```

1 nofun <- dcast(dcastfun, 省份 ~ 医院级别, value.var = "总收入")
# Aggregation function missing: defaulting to length
2 head(nofun)

```

```
3 havefun <- dcast(dcastfun, 省份 ~ 医院级别, value.var = "总收入", fun.aggregate =  
sum)  
4 head(havefun)
```

我们发现上面的结果截然不同，所以当你指定的 id 列不能保证每一行都是独一无二的时，这时就存在一个透视表的问题，你是要计数，还是要求和，还是要求均值，需要通过 `fun.aggregate` 指定一个透视函数，如果不指定就默认为计数函数，即 `length`，如第一句，它统计的不是北京医院二级医院的总收入，而是北京二级医院在表中出现的次数；第 3 行代码我们指定了透视函数为 `sum`，这样北京二级医院就按照求和的方式汇总了。

对于高级语言，如果你没有了解每一个参数的意义，那函数执行之后一定要检查结果是不是你要的。

### 2.2.3 关联合并表

在数据处理阶段，另外一个比较常见的工作就是关联，因为很多时候为了节省存储空间或方便查询，往往会将数据分表，比如一个表存储用户购买的产品信息，一个表存储客户的基本属性信息，如地址、性别等，如果将它们作为一个表存储，则可能会造成信息冗余，比如客户甲购买了  $n$  次物品，如果客户甲的属性信息也和购买信息存放在一张表内，他的地址、性别、年龄都要被重复  $n$  次存储，这样就造成了存储空间的浪费。我们不如将其分成两个表，一个表存储交易信息，一个存储客户的基本属性，然后两张表可以通过客户 id 关联，事实上大部分信息存储都是这么做的。

这样我们在数据分析时就经常需要将多个表关联，以形成一个完整的交易多方的信息描述数据表，就是所谓的数据关联，如下代码创建两个表。

- 创建关联表

```
1 df1 <- data.frame(CustomerId = c(1:3, 2, 3, 6), Product = c(rep("洗发水", 3),  
rep("兰州烧饼", 3)))  
2 df2 <- data.frame(CustomerId = c(2, 4, 5, 6), State = c(rep("广东", 2),  
rep("河南", 2)), nickname = c("风一样的男人", "一坨阳光", "文艺 213", "王宝强"))
```

上面我们创建了两个表，`df1` 存储每个客户的交易信息，即每个客户购买了什么商品，`df2` 存储了客户信息，包括地址和昵称，我们需要将这两个表关联起来，代码如下所示。

- merge 表关联

```
1 merge(x = df1, y = df2, by = "CustomerId", all = TRUE) #全连接  
2 merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE) #左连接  
3 merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE) #右连接  
4 merge(x = df1, y = df2, all = FALSE) #内连接  
5 merge(x = df1, y = df2, by = NULL) #笛卡儿集
```

`merge` 函数是 R 基础包所提供的的一个用于关联的函数，它几乎能完成绝大多数的表关联动作，上述代码中，第 1 行代码用于完成表的全连接，所谓全连接就是取表的并集，即 `df1` 和 `df2` 能关联的行都关联在一起，不能关联的行也一起给出在结果中，只不过缺失的数据使用 `NA` 表示，参数 `x` 用于指定第 1 个表，参数 `y` 用于指定第 2 个表，`by` 用于指定用于关联的列名称，也就是 id 列，你可以

通过向量的形式指定多个 id 列，如 `by = c("id1", "id2")`，`all` 参数等于 `T`，表示进行全连接，也就是 SQL 里面所说的 `full` 连接；第 2 行代码进行左连接，即保留第 1 个数据集 `df1` 的全部数据，能关联上的就用 `df2` 填充，`df2` 没有的 `id` 就用 `NA` 填充，总之结果中能查到第 1 个数据集的全部记录，这个动作使用 `all.x = TRUE` 指定；第 3 行代码进行右连接，正好相反，即保留 `df2` 中的所有 `id` 数据，使用 `all.y = TRUE` 指定；第 4 行代码进行内连接，所谓内连接就是取两个数据集 `id` 的交集，使用 `all = FALSE` 指定；第 5 行代码比较特殊，就是取两个数据集的 `id` 的笛卡儿集，也就是每一个 `df1` 中的 `id` 都要和 `df2` 中的所有 `id` 对应一次，严格来说这不是一种关联关系，因为它没有指定关联 `id`，而是将 `by` 参数设置为 `NULL`。

R 里面用于关联的函数多如牛毛，一般来说只需要掌握自己熟悉的几个就可以了，它们的唯一差别就是效率，只是有些关联函数比较快而已，比如 `dplyr` 包中的关联函数就比基础的 `merge` 函数快，如下所示。

- `dplyr` 包表关联

```
1 library(dplyr)
2 full_join(df1, df2, by = "CustomerId")
3 left_join(df1, df2)
4 inner_join(df1, df2)
5 anti_join(df1, df2)
```

上述代码中，第 1 行代码加载包；第 2 行代码进行全连接，同样通过 `by` 指定关联的 `id`；第 3 行代码进行左连接，`join` 函数和 `merge` 函数在未指定关联 `id` 的情况下默认使用数据框中所有列名称相同的列作为关联 `id` 进行关联，所以是否需要指定关联 `id` 一定要考虑清楚，不能图省事；第 4 行代码进行内连接；第 5 行代码比较有意思，`anti` 就是反的意思，就是取出 `df1` 中在 `df2` 中找不到的关联 `id` 数据，这一步可以通过 `left_join` 变通完成。

## 2.2.4 数据批处理：R 语言里最重要的一个函数家族：\*ply

之前我们讲过循环，简单讲解了向量计算，它可以代替循环进行比较搞笑的批量计算，试想计算数据框中的总收入加上 10，我们没必要进行循环计算这一列的每一个元素与 10 的和，只需要把这一列当成一个向量执行就可以了，这就是向量计算，但是问题来了，如果我要数据框中所有列都加上 10 或者求所有列的均值、中位数，又或者对它们调用更加复杂的函数，该怎么办？第一个想到的还是循环，但是 R 里面循环很慢，这时 \*ply 家族的函数就派上用场了，专门完成比向量更高层次的批量处理，如下所示。

- `apply` 函数

```
1 temp <- data.frame(a = rnorm(30, mean = 0), b = rnorm(30, 2), c = rnorm(30, 4))
2 apply(X = temp, MARGIN = 2, FUN = mean)
3 apply(temp, 2, sd)
4 apply(temp, 1, mean)
5 apply(temp, 2, function(x) length(x[x > 1]))
```

上述代码中，第 1 行代码生成一个数据框，它每一列的数都是从一个正态分布中抽取的样本，描述一个正态分布的样本集至少要包含 3 个要素：样本数、均值 mean、标准差 sd，函数 rnorm 的参数也主要包含这 3 个要素，主要用于随机产生一个指定特征的正态分布样本集，这里 a 列表示随机抽取一个包含 30 个样本、均值为 0、标准差为 1（默认是 1）的向量，然后赋值给 a 列，b 为随机抽取一个包含 30 个样本均值为 2 的正态分布向量，c 同样解释；为了验证抽样效果，我们可以求每一列的均值看看和设定的是否一样，这里不需要循环，只用 apply 函数即可，参数 X（大写）用于指定数据集，MARGIN 用于指定是对行计算还是对列计算，行用 1 表示，列用 2 表示，也可以同时进行行和列计算，最后一个参数 FUN 指定调用的函数，连起来解释即是把 temp 中的每一列丢给 mean 函数计算求均值，可以看到第 1 列的均值接近 0，第 2 列接近 2，第 3 列接近 4，和我们设定的一样；第 3 行代码对每一列求标准差，可以看到标准差都接近 1，和默认的标准差一样；第 4 行代码对每一行求均值；第 5 行代码使用了自编函数，这里是一个匿名函数，所谓匿名函数就是没有函数名，它统计 x 中大于 1 的个数，这句整体连起来是将 temp 的每一列丢给匿名函数，计算每一列中大于 1 的数值个数，有意思的是，“pply”家族大多数用于自编函数时进行一些批量的特殊操作。

除了对矩阵或者数据框进行操作以外，我们有时候还会用到对 list 的批量操作，这就需要用到 lapply 或者 sapply 函数了，如下所示。

- lapply 函数

```
1 temp <- list(a = rnorm(30, mean = 0), b = rnorm(10, 2), c = rnorm(20, 4))
2 lapply(X = temp, FUN = length)
3 length(temp)
4 myfun <- function(x) {
5   a <- median(x)
6   b <- mean(x)
7   return(c(a, b))
8 }
9 lapply(temp, myfun)
```

上述代码中，第 1 行代码创建了一个包含 3 个向量元素的 list；第 2 行代码使用 lapply 函数将 list 的每一个元素丢给 length 函数，统计每一个元素的长度，返回的结果是一个和 X 等长的 list，包含 3 个计数结果；第 3 行代码返回 temp 的长度为 3；第 4 行代码起编了一个自编函数 myfun，它计算 x 的中位数和均值，并将结果捆绑为向量返回；最后一行代码将 temp 应用于自编函数 myfun，返回计算结果，结果是一个 list，和 temp 等长，包含 3 个向量，每个向量包含两个元素，即 temp 中每个向量的中位数和均值。

有时候我们需要向函数传递多个参数，然后进行批量操作，如下所示。

- 向 lapply 函数传递多个参数

```
1 myfun2 <- function(x, y) {
2   a <- median(x + y)
3   b <- mean(x - y)
4   return(list(a, b))
5 }
```



```
6 b = 3
7 lapply(X = temp, myfun2, y = b)
```

上述代码中，第1行代码设计了一个自编函数 myfun2，它需要用户传递两个参数：x、y；最后一行代码使用 lapply 传递多个参数，只需要在 FUN 后面继续赋值 FUN 所需的参数就可以了，比如 myfun2 需要 y 参数，我们就在后面赋值 y = b，这样就可以完成传参了，表示 temp 中每个元素向量都要和 y 一起参与计算，是整体的 y，而不是 y 下面的每一个元素，因为 temp 是按元素被 lapply 函数传递，而 y 不是，y 是整体赋值。

- sapply 函数

```
1 sapply(X = temp, FUN = myfun, simplify = F)
2 sapply(X = temp, FUN = myfun, simplify = T)
```

sapply 函数就是 lapply 的一个简化版本，因为它添加了 simplify 参数而更名为 sapply，我们看到在 simplify 等于 F 时，它返回的结果和 lapply 一样是个 list，当 simplify 为 T 时它就将结果整理成一个矩阵，第10章会应用到很多 do.call 函数加 lapply 函数，我甚至已经忽略了 sapply 函数。

另外一个传递多个参数进行建模的函数为 mapply，这个函数很重要，一般在模型比较的交叉检验时经常用到，比如后面章节比较随机森林的树数对模型的影响时就用到了，它的一个作用就是避免多重循环，因为它和 lapply 的多参数传递不同，它是将多个参数的元素一一对应传递的，如下所示。

- mapply 函数

```
1 x <- 1:3
2 y <- c(4, 2, 2)
3 z <- c(1, 3, 4)
4 myfun3 <- function(x, y, z) {
5   m <- y + z*x
6 }
7 mapply(myfun3, x, y, z)
8 lapply(x, myfun3, y = y, z = z)
```

上述代码中，前三行代码创建了3个向量；第4行代码创建了一个自编函数 myfun3；mapply 函数将 x、y、z 的元素一一对应地传递给 myfun3，返回的变量是一个向量，包含了3个元素，因为 x、y、z 的元素共产生了3次一一对应的关系，看到了吗，如果你将要使用一个三重循环就可以使用这种方法避免，速度快很多，具体用例请参看第10章；而 lapply 返回的结果就比较复杂了，所以它们的传参逻辑不是一回事。

除了上面讲到的函数以外，“\*ply”家族还有很多变种，比如 ddply、vapply、rapply 等，大同小异，根本没必要记忆，因为上面的函数足够完成它们的工作，比如 tapply 函数比较像分组函数，可以使用之前的透视表函数替代它，更加方便。其实一旦你真的弄懂之后，就会发现很多函数都是“浮云”，我仅仅用到两个“\*ply”家族的函数 lapply 和 mapply，其他的都被其他函数替代掉了。

## 2.3 一招完成数据探索报告

以上我们讲了大量的数据清洗工作的用例，数据清洗完成之后就进入了数据探索阶段。项目紧张时，数据探索常常被忽视，需要注意的是数据探索是数据挖掘工程师将数据和业务问题连接的必经之路，良好的数据探索除了快速了解数据的情况以外，还可以为充分思考自己的挖掘方案预留时间，从而碰撞出更加神奇的想法。

那么问题来了，数据探索都探索什么？这是一个很宽泛的问题（请把“很”拉长音），因为探索的深度不同可能探索的角度也花样繁多，从这个方面说探索数据就是玩数据，以将数据玩透为目的，但是有时候工作不是玩，玩透了就要对数据进行描述，这就是所谓的描述性统计，专业的工作基本上是批量化的程式性工作，那么我们就必须给出几个探索数据的点。

### 常见数据探索的角度

- (1) 数据结构什么样，包含哪些变量
- (2) 变量分别为什么类型
- (3) 变量包括多少非重复值
- (4) 变量的均值
- (5) 变量的重要百分位点及最大值最小值
- (6) 变量包含多少缺失值

初步的数据探索至少包含以上几点，如果更加深入可能还要探索变量之间的相关性等，我们后面章节详细讲，但是初步数据探索至少应该包括以上提出的对数据特征表述的关键点。下面一一讲述。

### • 数据抽样

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招  
招都是利器/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)  
2 sampleseq <- sample(1:length(hospital[,1]), size = 1000, replace = F)  
3 temp <- hospital[sampleseq,]
```

在进行数据建模或数据探索之前，如果数据量比较大，可能需要先对数据抽样，我们刚开始写一个项目的代码时，需要快速把代码执行一遍，以测试代码是否能够正常工作，但是如果使用全数据可能每一步都要花费大量的执行时间，这时数据抽样就是必须的了，抽样能节省代码测试的时间。上述代码中，第 1 行代码读取数据；第 2 行代码的 sample 函数用于随机抽样，它可以在一个序列中随机抽取指定量的样本，1:length(hospital[,1])用于生成一个和 hospital 行数等长的序列，然后从中抽出 1000 个 (size)，抽样方式为非放回式，所谓非放回式即抽了之后不再放回到总体内，这样就生成了一个行编号样本；第 3 行代码使用这个行编号样本去 hospital 里面提取相应的行，就完成了数据抽样。代码测试时进行数据抽样绝对是一个提高工作效率的好习惯。

### • 单变量探索

```
1 str(hospital)  
2 summary(hospital$诊疗人次)
```

```
3 quantile(hospital$诊疗人次, probs = seq(0, 1, 0.10), na.rm = T)
4 length(unique(hospital$省份))
```

上述代码中，第1行代码的 `str` 函数用于查看数据的结果，大概了解下数据包括哪些变量、变量是什么类型等；第2行代码的 `summary` 函数用于计算变量常见的统计量，比如计算了诊疗人次的最小值、下百分位点、中位数、均值、上百分位点、最大值和趋势值数量等统计量，非常方便；第3行代码的 `quantile` 函数用于计算数据的百分位数，参数 `probs` 设置计算的百分位点，比如你可以设置为 `probs = c(0.2, 0.4)`，即计算 20% 百分位数和 40% 百分位数，这里使用 `seq` 函数产生一个从 0 开始到 1 结束公差为 0.1 的序列，这就表示计算诊疗人每隔 10 个点的百分位数，`na.rm` 用于设定是否移除缺失值；最后一行代码统计省份的个数，首先使用 `unique` 函数对省份进行去重，上面的函数基本上完成了大部分的数据探索任务，但是我们需要出一份数据探索报告，就要进行批量操作。

数据探索很重要，但其报告不一定非得要做成高大上的 PPT、PDF，原因很简单，当数据表比较复杂时不方便查询，所以最好还是做成 CSV 格式，但如果要拿出去给客户看还是挑选极个别能说明问题的探索项目进行可视化比较好，我们先出具一份 CSV 格式的探索报告。

- 批量数据探索：数值型

```
1 library(Hmisc)
2 isnum <- sapply(hospital, is.numeric)
3 temp <- hospital[,isnum]
4 numvars <- describe(temp)
5 numvars
6 str(numvars)
7 str(numvars$床位数)
8 numvars$床位数[["counts"]]
9 temp <- sapply(numvars, "[", "counts")
10 numdescribe <- t(temp)
```

Hmisc 包的 `describe` 函数能完成大部分的数据探索，但是 `describe` 函数对数值型和字符型变量计算的角度不一样，所以我们需要将数值型变量和字符型变量分开分析。上述代码中，第1行代码加载包 Hmisc；第2行代码使用 `sapply` 函数判断 `hospital` 中每一列是否是数值变量，是返回真，不是返回假；第3行代码使用上一步产生的序列将 `hospital` 中数据类型为数值的列筛选出来并赋值给 `temp`；第4行代码使用 `describe` 函数对 `temp` 中的变量进行探索；第5行代码查看结果；第6行代码的 `str` 函数查看探索结果，结果是一个 list 包含 3 个元素，分别是床位数、诊疗人次、总收入；第7行代码的 `numvars` 的每个元素又是一个 list，比如床位数是一个 list，它包含 6 个元素，而我们需要的数据存储在它的 `counts` 元素中；第8行代码提取 `numvars` 下的床位数的 `counts` 元素，这是一个 list 嵌套 list 的数据结构，这样我们就得到了对床位数的描述；第9行代码使用 `sapply` 函数批量提取，表示将 `numvars` 的每一个元素传递给函数 “[”（它是提取 list 元素的函数，前面已经说过），并补充了函数 “[” 的参数 `counts`，这样我们就提取了每个变量的描述性统计量，也就是探索结果 `temp`；第10行代码将探索结果行转列进行转置一下就可以了，使用的是转置函数 `t`，这样数值型变量的探索就完成了，其结果包含了变量数值个数、缺失值个数、去重复值个数、均值、连续性（参看帮助文

档解释)及百分位数。

- 批量数据探索：字符型

```
1 temp <- hospital[,!isnum]
2 charvars <- describe(temp)
3 chardescribe <- t(sapply(charvars , "[", "counts"))
```

上述代码中，第 1 行代码将上一步得出的 isnum 逻辑序列变为非，就可以提取非数值型变量了，并将提取结果存储在 temp 中；第 2 行代码进行数据探索；第 3 行代码提取探索结果，并使用转置函数转置，大家看到我将上一节的提取和转置写成了一句代码，虽然代码简洁了，但是请记住减少代码行数经常会拖慢你的代码速度，增加代码行数往往会吃掉你的大部分内存，除非你经常移除中间变量，我建议不要轻易写复杂的代码，速度慢、不易读，经过上面的整理我们就将字符型变量的探索整理完成了。

- 批量探索：出报告

```
1 varname <- c(row.names(numdescribe), row.names(chardescribe))
2 numdescribe <- data.frame(numdescribe, stringsAsFactors = F)
3 numdescribe <- data.frame(lapply(numdescribe, as.numeric))
4 chardescribe <- data.frame(chardescribe)
5 library(dplyr)
6 hospitaldatapofile <- bind_rows(numdescribe, chardescribe)
7 hospitaldatapofile <- data.frame(varname, hospitaldatapofile)
8 write.csv(hospitaldatapofile, "hospitaldatapofile.csv")
```

我们需要将两个探索报告合并在一起，但在合并之前首先要进行一些调整转化，比如因为原始变量的名称被用来表示 numdescribe、chardescribe 两个矩阵的行名称，所以首先要提取出来。上述代码中，第 1 行代码提取了两个矩阵的行名称也就是变量名称，记住名称提取的先后顺序要和后面合并时的先后顺序一致，均为先 numdescribe 后 chardescribe；第 2 行代码将 numdescribe 转化为数据框；第 3 行代码因为 numdescribe 的每一列是字符，但实际上它们应该是数值型，所以需要将它们转化为数值，使用 lapply 函数将字符型变量转化为数值型，然后外面的 data.frame 将转换的结果再转化为数据框，这样 numdescribe 的数据就变成数值型了；第 4 行代码将 chardescribe 转化为数据框；第 6 行代码将两个数据框按行捆绑在一起，使用的是 dplyr 包的 bind\_rows 函数，因为它能将两个列数不同的数据框按列捆绑，缺失的列会被填充为 NA，一般情况下两个数据框的列数不一样需要补全后才能按行捆绑，所以 bind\_rows 函数真是太方便了；第 7 行代码将变量名和探索结果 hospitaldatapofile 捆绑在一起；第 8 行代码将结果输出为 CSV 文件就完成了了一份简单的数据探索报告，存放在当前目录下，可以通过 getwd()查看当前工作目录。

探索报告首先要仔细地看一下，比如如果性别出现了 3 个非重复值，那就要了解一下有哪 3 种，方便领导问到时解答，如果一份报告你自己都不看，那出这份报告的意义何在呢？除了出 CSV 版的报告还可以出 PDF 版的探索报告，你先要下载安装 ProTeXt-3.1.5-033015，下载地址为：<http://mirror.neu.edu.cn/CTAN/systems/win32/protext/>，提取安装 MiKTeX，或者直接通过百度搜索 MiKTeX。

仍然是 Hmisc 包里的 describe 函数，能够快速识别变量的类型（字符、因子、分类、哑变量、离散变量、连续变量），根据变量类型计算一些简洁的统计特征，最后使用 LaTeX 为报告添加频率柱状图，一份简洁而不简单的报告就完成了，如下所示。

- 出具 PDF 版本数据探索报告

```
1 hospitaldesc <- describe(hospital, size = "normalsize")
2 x <- latex(hospitaldesc, file = "describe.tex")
3 text2 <- "\\documentclass{article}\\n\\usepackage{relsize,setspace}\\n\\begin{document}\\n\\input{describe.tex} \\n\\end{document}"
4 cat(text2)
5 print(text2)
6 cat(text2, file = "Hmisc_describe_report.tex")
7 library(tools)
8 texi2dvi("Hmisc_describe_report.tex", pdf = TRUE)
9 getwd()
```

上述代码中，第 1 行代码完成数据探索；第 2 行代码将结果转化为 LaTeX 对象，并输出为 describe.tex；第 3 行代码是一段 LaTeX 的语法，即最流行的排版软件，其大概意思是一个段通用的标志语言讲明了需要排版文件的名称；第 4 行代码的 cat 函数本是打印函数，比如打印字符串等，它会按照文本标记语言打印输出结果，比如“\”在 R 中是逃逸符，专门用于输出一些比较特殊的符号，比如你要输出“\”就要输入两个“\\”才可以正常输出，另外 text2 中还有换行符“\n”，它在 cat 中表示换行，可以比对输出结果；而第 5 行代码的 print 函数则不会进行标记语言的转换，就是输入什么就按照字符串输出，这一点是它们的区别；第 6 行代码将 text2 使用 cat 函数输出写入到 Hmisc\_describe\_report.tex 文件中；第 7 行代码加载 tools 包，R 里面用于管理 R 包及其文档的包；第 8 行代码的 texi2dvi 函数将 tex 对象转化为 PDF 输出，存放在当前工作目录下；第 9 行代码通过 getwd 函数查看当前目录。

这样就对 mtcars 数据集生成一个完整的探索文档了，你可以去当前的工作目录里面查找 PDF 格式的报告文档。PDF 版本的报告具有可视化的内容，一般比较适合给领导看。数据分析师使用便于操作和筛选的 CSV 版本会更加得心应手。两种方式相比之下我更喜欢第一种导出 CSV 文件的方式，因为这种方式很简洁方便，当然如果领导要求你出具一份探索报告，还是使用融合了图形的 PDF 方式比较好。

## 2.4 拯救你的很多时候是基础理论

其实数据挖掘工程师一般是从两个群体里面而来的，其一是由数学、统计学、应用科学（如生物统计学）相关的学科转行而来的，这群人统计学基础扎实，比较注重分析过程的统计基础和算法，但是代码的速度和简洁程度差些，他们一般选择 R 和 Matlab 之类的分析工具；其二是由程序员转行而来的，这类人代码质量较高，但是分析过程的统计学基础较差，他们一般选择 Python 之类的分析

工具。但可笑的是，总体而言，这些人都会逐渐忘掉原来所学的统计基础，有时候记住了很多高大上的算法却忘记了基础的统计分析，比如分析哪一种人的收入较高时，或者哪一种基因导致的转氨酶不正常表达时，竟然无法对应到基础的均值比较或方差分析，评价算法的效果时竟然不能自己独立构建评价指标，有时候即使做了评比，也不具有统计意义，甚至有人滥用“显著”这种具有特殊含义的统计字眼。

### 2.4.1 参数检验及非参检验

参数检验最常见的是均值比较，本书中涉及的统计学问题一般是如果不讲清楚就无法继续的统计学知识，而均值比较就是这一类，它的重要性几乎会纠缠每一个数据挖掘工程师的一生，也会贯穿一个项目周期。“比较”一般是通过参数检验和非参检验进行的，而均值比较就是最常见的参数检验，比如你要比较两个模型的效果，肯定要进行准确率或者 mae、mse、rmse、mape 等评价指标的均值的比较，单纯的一次建模和测试就进行比较是没有统计意义的。

比如我们要比较同时期的人群收入是否存在差异，比如我们要分析三级医院和二级医院的收入差异，这些都要用到均值比较。

均值比较中最常用的是  $t$  检验(样本量小于 30 用  $z$  检验)， $t$  检验要求数据符合正态分布的假设，所以在进行  $t$  检验之前首先要进行正态分布的检验，正态分布的检验分为两类；一种是公式法，如 Shapiro-Wilk、Anderson-Darling、Kolmogorov-Smirnov 等检验方法；另外一种是图形法，如 qq 图、频度直方图等。作为一个经常要求分析能够自动化进行的业内人士，我更倾向于使用第一种，符合正态分布才能进行  $t$  检验，但是这个两步法并非没有问题，详情参看 *Is normality testing 'essentially useless'*。

下面进行一个单样本的均值检验，比如我们想知道北京的二级医院平均年收入是否高于 1 亿元，这就用到了均值检验，这类问题仅包含一个样本和具体值的比较，所以称为单样本  $t$  检验。首先要检验北京的医院收入是否符合正态分布，然后再进行  $t$  检验。

- qq 图检验正态分布

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招  
招都是利器/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)  
2 bjhospital <- hospital[hospital$省份 == "北京" & hospital$医院级别 == "二  
级",]  
3 summary(bjhospital$总收入)  
#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.     
#      0  27250   84790 125500 181200  548200   
4 qqnorm(bjhospital$总收入)  
5 qqline(bjhospital$总收入, col = 2)  
# 标准的正态分布  
6 temp <- rnorm(1000, mean = 2)  
7 qqnorm(temp)  
8 qqline(temp, col = 2)
```

图形检验正态分布通常在 PPT 展示时比较常用，方法有频率图、密度图、qq 图，它们有共同的优点，比较直观。上述代码中，第 1 行代码读取医院的数据；第 2 行代码筛选出北京市的二级医院，“&”在 R 语言里表示且的关系；第 3 行代码使用 summary 看一下北京二级医院总收入的统计摘要，可以看到北京医院的平均年收入为 1.2 亿元，那我们是否现在可以说北京二级医院总收入平均显著大于 1 亿元呢？显然这个时候做出这样的表述有点不负责任，且显得缺乏专业素养，因为还没进行统计意义上的比较；第 4 行代码使用 qqnorm 绘制 qq 图，qq 是 quantile 的缩写，用于检验样本的理论百分位数分布于实际百分位数是否相符，如果相符，则纵轴是数据的样本分位数，横轴是正态分布的理论分位数，沿 45 度的参考线；第 5 行代码的 qqline 函数为 qq 图添加标准的拟合线，如果数据是正态分布，则散点应该落在直线上，即实际分位数与理论分位数之间偏差很小，反之，如果偏离较大，则说明数据并非来自正态总体，但是我们看到北京二级医院的总收入并没有与理论上完全相符，显然有些扭曲，至少不是标准的正态分布，这时图形法的缺点就出现了，即依据图形法判断实在太过主观，一旦和标准不太一样时，只能根据个人经验判断；第 6~8 行代码绘制了一张标准的 qq 图，首先我们使用 rnorm 产生了一个均值为 2 的标准分布，第 7 行代码绘制 qq 图，第 8 行代码添加拟合线，我们看到标准的正态分布基本上沿线分布，看了这幅图等于给自己一个标准的印象，如图 2-3 所示。

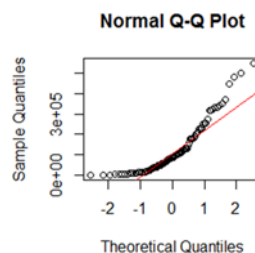


图 2-3

#### • 正态分布检验

```
1 shapiro.test(bjhospital$总收入)
# Shapiro-Wilk normality test
#
# data: bjhospital$总收入
# W = 0.84748, p-value = 2.334e-08
2 ks.test(bjhospital$总收入, "pnorm")
# One-sample Kolmogorov-Smirnov test
#
# data: bjhospital$总收入
# D = 0.98925, p-value < 2.2e-16
# alternative hypothesis: two-sided
```

常见的检验正态分布的方法之前说过，但是我更加倾向于选择 Kolmogorov-Smirnov 方法，Shapiro-Wilk 方法对样本大小的依赖性很大，一般样本量小于 50，这也是 shapiro.test 函数将样本量限制在 5000 个以内的原因，如果样本比较大，Shapiro-Wilk 很容易拒绝原假设得出不符合正态分布的结论，比如上面第 1 行代码使用 shapiro.test 检查总收入的正态性，看到 p-value 远远小于 0.05 甚至远远低于 0.01，所以拒绝原假设，北京二级医院的总收入不符合正态分布，如果 p-value 远远大于 0.05 则表示不能拒绝原假设，也就是说不能够否定数据属于正态分布，很遗憾北京二级医院的收入不符合正态分布，因为 ks.test 函数的检验结果也支持这种结论。

但是也不要太担心，其实有时候在实战操作中如果样本量很大，就直接认为符合正态分布了，

另外，即使原假设被拒绝了，也可以根据数据背景讨论，数据可能受某些异常值的影响，偏离了正态分布，可以根据上面讲到分析异常值的方法将异常值剔除之后再检验，实在通不过还可以进行非参检验，这里我们先假设数据符合正态分布，继续往下一步走。

- 单样本  $t$  检验

```
t.test(x = bjhospital$总收入, alternative = "greater", mu = 100000)
# One Sample t-test
#
# data: bjhospital$总收入
# t = 1.9485, df = 92, p-value = 0.0272
# alternative hypothesis: true mean is greater than 1e+05
# 95 percent confidence interval:
# 103762 Inf
# sample estimates:
# mean of x
# 125546.7
```

上述代码中，第 1 行代码使用 `t.test` 函数进行均值比较，比较北京二级医院收入的均值是否大于 1 亿元，第 1 个参数用于指定样本数据，`alternative` 指定检验类型，这里指定单尾检验，我们看到  $t$  值为 1.9,  $p$ -value 小于 0.05, 我们可以认为拒绝原假设, 认为北京二级医院平均收入显著大于 1 亿元。

除了单样本的均值检验以外，我们还会遇到双样本的均值检验，在进行  $t$  检验之前一般认为两个样本来自于方差相同的正态总体，比如我们检验北京医院的平均收入和河南医院的平均年收入。如下所示。

- 双样本  $t$  检验

```
1 bjhospital <- hospital[hospital$省份 == "北京", "总收入"]
2 hnhospital <- hospital[hospital$省份 == "河南", "总收入"]
3 shapiro.test(bjhospital)
# Shapiro-Wilk normality test
#
# data: bjhospital
# W = 0.40661, p-value < 2.2e-16
4 shapiro.test(hnhospital)
# Shapiro-Wilk normality test
#
# data: hnhospital
# W = 0.29183, p-value < 2.2e-16
5 result <- t.test(x = bjhospital, y = hnhospital, var.equal = FALSE)
# Welch Two Sample t-test
#
# data: bjhospital and hnhospital
# t = 6.0533, df = 579.38, p-value = 2.55e-09
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
```



```
# 39078.56 76617.38
# sample estimates:
# mean of x mean of y
# 79660.77 21812.80
6 result
7 result$statistic
8 result$p.value
9 result$conf.int
```

上述代码中，第1行代码提取北京医院的数据；第2行代码提取河南医院的数据；第3~4行代码检验样本是否符合正态分布，结果两个省市的医院收入都不符合正态分布，按道理下面就不能轻易进行  $t$  检验，但是我们这里为了演示，继续进行；第5行代码进行  $t$  检验，同时设置了方差不相等，如果你的数据方差相等就将此项设置为 T，另外需要注意的是 `t.test` 还有一个非常重要的参数，`paired = FALSE`，用于指定是否进行配对数据检验，如果设为 T，则表示数据是一一配对的，比如对同一批病人一个1月转氨酶与2月转氨酶是否存在差异，就可以使用配对数据检验，因为检验的是同一个样本的先后差异；第6行代码查看结果，看到北京医院的平均年收入为7966万元，河南的医院平均年收入为2181万元，它们之间差值的95%的置信区间为[39078.56, 76617.38]，因为  $p$ -value 远远小于0.01，所以拒绝原假设，认为两个省份的医院收入存在极显著差异；有时候我们需要提取结果的一些统计量存储备用，比如第7~9行代码分别提取了  $t$  值、 $p$ value 和差异置信区间。

除了均值检验以外，我们可能还会进行非参检验，非参检验一般是在数据不符合参数检验要求时进行，比如上一步我们的数据不符合正态分布的要求，就可以进行非参检验，这里重点演示一下秩和检验和卡方检验。

秩和检验适用于上一步中检验不符合正态分布的情况，比如我们比较北京二级医院平均收入是否大于1亿元，但样本不符合正态分布，就可以使用秩和检验，秩和检验是进行中位数之间的比较的，如下所示。

#### • 单样本秩和检验

```
1 bjhospital <- hospital[hospital$省份 == "北京" & hospital$医院级别 == "二
级", ]
2 summary(bjhospital$总收入)
#      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#         0  27250   84790 125500 181200 548200
3 wilcox.test(bjhospital$总收入, alternative = "greater", mu = 100000)
#      Wilcoxon signed rank test with continuity correction
#
# data:  bjhospital$总收入
# V = 2297, p-value = 0.3353
# alternative hypothesis: true location is greater than 1e+05
```

上述代码中，第1行代码提取北京市二级医院的收入；第2行代码查看统计摘要，可以看到北京市二级医院的收入中位数为8500万元左右，是不是存在大于1亿元的可能呢？第3行代码我们使用 `wilcox.test` 函数进行了 `wilcox` 秩和检验，发现结果  $p$ -value 等于0.33，远大于0.05，所以不认为北

京二级医院收入的中位数大于 1 亿元。

- 秩和检验

```
1 bjhospital <- hospital[hospital$省份 == "北京", "总收入"]
2 hnhospital <- hospital[hospital$省份 == "河南", "总收入"]
3 wilcox.test(x = bjhospital, y = hnhospital)
#   Wilcoxon rank sum test with continuity correction
#
# data:  bjhospital and hnhospital
# W = 347210, p-value = 0.0005514
# alternative hypothesis: true location shift is not equal to 0
4 wilcox.test(x = bjhospital, y = hnhospital, alternative = "greater")
#   Wilcoxon rank sum test with continuity correction
#
# data:  bjhospital and hnhospital
# W = 347210, p-value = 0.0002757
# alternative hypothesis: true location shift is greater than 0
```

上述代码中，第 1 行和第 2 行代码提取北京市和河南省的医院收入；第 3 行代码的 `wilcox.test` 函数检验两省市的医院收入中位数是否存在差异，`p-value` 远远小于 0.01，更不用说 95% 的置信度，可以确认两省市的医院收入中位数存在差异；第 4 行代码的 `wilcox.test` 函数检验北京市医院收入的中位数是否大于河南省的，结果 `p-value` 远远小于 0.01，接受备择假设认为北京市医院收入的中位数极显著大于河南省医院收入中位数。

另外我们经常需要检验两个变量是否相关，比如医院收入是否和医院的床位数相关，这时就用到了 `cor.test` 函数，用于比较相关性。如下所示。

- 相关性检验

```
1 cor.test(x = hospital$总收入, y = hospital$床位数, method = "pearson")
#   Pearson's product-moment correlation
#
# data:  hospital$总收入 and hospital$床位数
# t = 190.48, df = 19169, p-value < 2.2e-16
# alternative hypothesis: true correlation is not equal to 0
# 95 percent confidence interval:
#  0.8039421 0.8137304
# sample estimates:
#      cor
# 0.8088923
```

`cor.test` 函数用于计算两个变量的相关性，结果发现总收入和床位数存在非常明显的正相关，相关系数为 0.809，`p-value` 远远小于 0.01，接受备择假设，认为两个变量之间存在极显著的相关性，`cor.test` 函数的参数 `method` 用于指定计算相关系数的方法，如果是连续型变量用 `pearson`，如果是等级相关的变量请用 `spearman`，即秩相关，比如比较两个排名结果是否存在差异等，就可以转化成两个排名的 `spearman` 相关系数的检验，排名章节将详细介绍。与之相比，我们更需要尽快了解卡方检验，比

如我们要检验样本是否符合某种概率分布，就要用到卡方检验了。

例如，我们分析大麦杂交后形状比例是否符合“无芒：长芒：短芒=9：3：4”，实际观测值 435：85：146；则可以进行如下检验。

- 卡方检验

```
1 prob <- c(9/16,3/16,4/16)
2 x <- c(435, 85, 146)
3 chisq.test(x, p = prob)
# Chi-squared test for given probabilities
#
# data: x
# X-squared = 24.987, df = 2, p-value = 3.751e-06
```

上述代码中，第1行代码生成理论概率；第2行代码创建对象 x，按照概率类别的顺序输入数值；第3行代码使用 chisq.test 函数进行卡方检验，结果 p-value 远远小于 0.01，拒绝原假设，认为真实值和概率分布存在非常明显的差异。

另外卡方检验通常用于检验两个现象是否是独立的，比如吸烟和患肺癌是否是独立现象，如下例。

- 卡方独立性检验

```
1 x <- matrix(c(90,10,12,31),nrow=2)
2 colnames(x) <- c("患癌", "未患癌")
3 rownames(x) <- c("吸烟", "不吸烟")
4 chisq.test(x)
# Fisher's Exact Test for Count Data
#
# data: x
# p-value = 2.31e-13
# alternative hypothesis: true odds ratio is not equal to 1
# 95 percent confidence interval:
# 8.386529 66.055954
# sample estimates:
# odds ratio
# 22.44285
5 fisher.test(x)
# Fisher's Exact Test for Count Data
#
# data: x
# p-value = 2.31e-13
# alternative hypothesis: true odds ratio is not equal to 1
# 95 percent confidence interval:
# 8.386529 66.055954
# sample estimates:
# odds ratio
```

```
# 22.44285
```

上述代码中，第 1 行代码创建一个样本统计矩阵；第 2 行代码修改矩阵列名；第 3 行代码修改矩阵行名称；第 4 行代码的 `chisq.test` 函数进行卡方独立性检验，结果 `p-value` 远远小于 0.01，认为二者之间存在非常强烈的联系；有时 `chisq` 检验的敏感度较低，可以使用费雪检验（`fisher.test`），结果同样表明二者之间存在非常明显的相关性（见第 5 行代码）。

基本上到这里我们已经将统计学的基础知识参数检验和非参检验过了一遍，本书并非完全的统计学教参，疏漏难免，但是可以肯定的是，数据挖掘之道就是要遵循基本的统计学，脱离了统计逻辑的报告和过程都是离经叛道之举，陷自己于不专业的境地，比如没有进行合格的统计比较就乱用“显著”和“极显著”这样的字眼等。

## 2.4.2 学了很多算法却忘了方差分析

了解一种统计知识，首先要知道它是用来解决什么问题的，然后了解它有哪些指标及各个指标的阈值，最后才是它的原理，方差分析的目的就是解决多个因素或多个样本之间均值比较的问题。换句不太准确的话说，即它和多重比较加一起就相当于均值比较的批量版本。比如我们比较不同级别之间的医院收入是否存在差异，就可以使用方差分析来解决这个问题，如下所示。

- 方差齐性检验

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 2 章数据探索招
招都是利器/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)
2 bartlett.test(总收入 ~ 医院级别, data = hospital)
# Bartlett test of homogeneity of variances
#
# data: 总收入 by 医院级别
# Bartlett's K-squared = 44917, df = 3, p-value < 2.2e-16
```

在进行方差分析之前首先要进行方差齐性检验，这是方差分析的一个前提假设，上述第 2 行代码使用 `bartlett.test` 函数检验方差齐性，第 1 个参数是一个表达式，前面是目标变量后者是影响因素，如果要进行多因素可以在后面以“+”号添加因素，`p-value` 值远远小于 0.01，所以拒绝原假设，认为方差不齐，也就是表示不可比较或存在差异，那就没必要进行下一步的方差分析了，这里我们暂时假设方差 `p-value` 大于 0.05，认为方差具有同质性，可以进行下一步方差分析，然后再讲解方差不齐的情况，如图 2-4 所示为统计学中方差分析的流程。

- 方差分析

```
1 aov1 <- aov(总收入 ~ 医院级别, data = hospital)
2 summary(aov1)
#              Df    Sum Sq   Mean Sq F value Pr(>F)
# 医院级别      3 6.836e+13  2.279e+13   3695 <2e-16 ***
# Residuals 19963 1.231e+14  6.166e+09
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

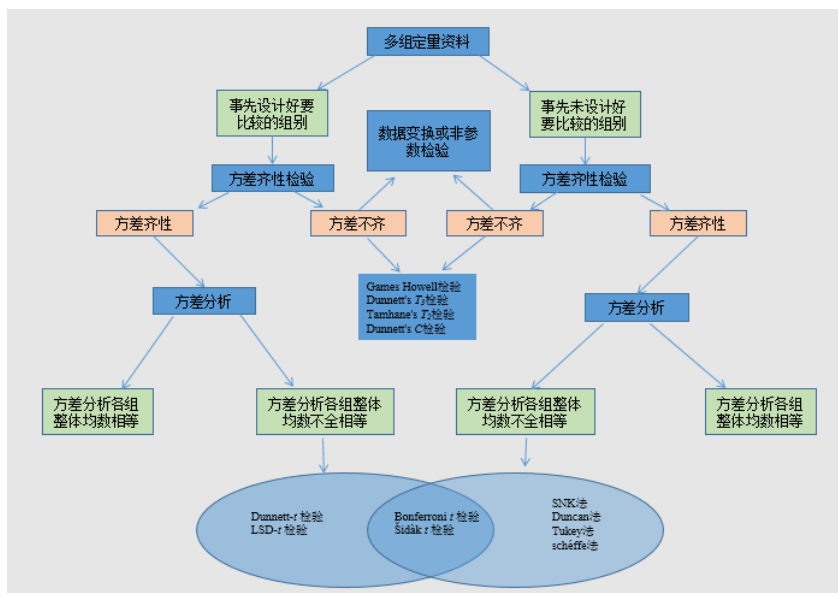


图 2-4

上述代码中，第 1 行代码使用 `aov` 函数进行方差分析，第 1 个参数用于指定方差分析的表达式，前面是目标变量，后面是影响因素，如果要进行多因素方差分析可以将表达式写为“总收入 ~ 医院级别 + 医院等次”，即表示分析医院级别、医院等次对医院收入的影响；第 2 行代码使用 `summary` 函数查看结果摘要表明  $\text{Pr}( > F )$  为  $2\text{e-}16$ ，远远小于 0.01，认为不同级别的医院收入存在极显著的差异，那么这里只是表明结果整体之间存在差异，如果进一步问哪些组之间存在差异呢？这里就用到了多重比较，如下所示。

- 多重比较

```
1 TukeyHSD(aov1)
#   Tukey multiple comparisons of means
#     95% family-wise confidence level
#
# Fit: aov(formula = 总收入 ~ 医院级别, data = hospital)
#
# $医院级别
#
#           diff           lwr           upr      p adj
# 三级一二级 221433.5245 215099.445 227767.604 0.0000000
# 未评级一二级 -29084.7746 -32522.206 -25647.343 0.0000000
# 一级一二级 -29388.7118 -33151.237 -25626.187 0.0000000
# 未评级一三级 -250518.2991 -256837.677 -244198.921 0.0000000
# 一级一三级 -250822.2362 -257324.174 -244320.298 0.0000000
# 一级一未评级 -303.9372 -4041.659 3433.785 0.9967776
```

TukeyHSD 函数用于分析方差分析的结果，并进行组与组之间的比较，如果  $p_{adj}$  小于 0.05，则表示两组之间差异显著，如果小于 0.01，则表示差异极显著，如果大于 0.05，则表示差异不显著；我们看到除了一级对未评级这一组之间的收入均值差异不显著以外，其他都达到了极显著的水平。

不要忘了我们检验方差齐性的结果是异质，也就是说不能进行普通的方差分析，R 里面处理方差异质的函数有很多，比如以下代码。

- 方差不齐的分析

```
1 oneway.test(总收入 ~ 医院级别, data = hospital, na.action=na.omit,
var.equal=FALSE)
# One-way analysis of means (not assuming equal variances)
#
# data: 总收入 and 医院级别
# F = 997.98, num df = 3.0, denom df = 4693.7, p-value < 2.2e-16
```

处理方差异质的方法之一就是使用 Welch correction 调整  $f$  检验的自由度，oneway.test 函数就是使用此方法完成方差异质时的方差分析的，结果同样表明不同级别的医院其收入存在极显著的差异。

### 2.4.3 多因素方差分析及协方差作用

除了进行单因素方差分析以外，我们还可以进行多因素方差分析，上面已经说过，只需要在表达式的后面添加变量就可以了，但是大家可能看到我们截至现在进行的分析都是因素或者因子性变量对目标变量的分隔，除了 cor.test。

- 多因素方差分析

```
1 aov2 <- aov(总收入 ~ 医院级别 + 医院等次, data = hospital)
2 summary(aov2)
#           Df    Sum Sq   Mean Sq F value Pr(>F)
# 医院级别      3 6.836e+13  2.279e+13  3835.1 <2e-16 ***
# 医院等次      3 4.511e+12  1.504e+12   253.1 <2e-16 ***
# Residuals 19960 1.186e+14  5.941e+09
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
3 aov3 <- aov(总收入 ~ 医院级别 * 医院等次, data = hospital)
4 summary(aov3)
#           Df    Sum Sq   Mean Sq F value Pr(>F)
# 医院级别      3 6.836e+13  2.279e+13  4195.7 <2e-16 ***
# 医院等次      3 4.511e+12  1.504e+12   276.9 <2e-16 ***
# 医院级别:医院等次  9 1.024e+13  1.138e+12   209.5 <2e-16 ***
# Residuals 19951 1.083e+14  5.431e+09
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

上述代码中，第 1 行代码进行多因素方差分析；第 2 行代码的结果中可以看到不同医院级别和等次之间收入均值的差异极显著， $Pr(>F)$  都小于 0.01；多因素进行方差分析时除了会产生主效应，因

素之间的交叉也可能对目标变量造成影响，这种作用称为相互作用，也可通过方差分析；第3行代码分析医院级别和等次对收入的影响，同时分析它们之间的相互作用对收入均值的影响，只不过把表达式中的加号换成了\*而已；第4行代码查看结果发现医院级别和等次的相互作用（医院级别:医院等次）对收入的影响也达到了极显著的水平。

上面我们说过，方差分析一般研究因子变量对目标变量的影响，因子变量一般在实验中可控，比如施药、施肥等，但是除了分类的因子变量外，还有连续型变量对目标变量的影响，分析这类问题时一般使用回归或者相关性检验，连续型变量一般是不可控的，比如实验对象的体重、身高、人群特征、经济指标等，至少不那么容易控制，比如施药的浓度，你很难说自己施药的浓度非常准确地为0.00001克每毫升，而协方差分析就是建立在方差分析和回归分析基础之上的一种用于分析可控的因子变量和不可控的数值变量对目标变量的影响，你是不是发现我们的关键词从“比较”变成了“影响”，没错，比较恰恰是对影响力的比较。

协方差分析一个特别的例子就是分析不同类型的饲料对猪体重的影响，饲料类型是因子变量可以控制，但是如果不考虑实验猪本身的条件，如初始体重，也许结果就不那么靠谱了，但是猪的初始体重是连续变量，不那么好控制，如果我们要同时分析它们对猪体重增量的影响就要使用协方差分析，这些内容在统计学教材中很普遍，我们这里继续以医院收入数据演示协方差分析。

- 协方差分析

```
1 hospital <- hospital[hospital$医院级别 == "二级"|hospital$医院级别 == "三级", ]
2 mod.f <- lm(总收入 ~ 医院级别, data = hospital)
3 anova(mod.f)
# Analysis of Variance Table
#
# Response: 总收入
#           Df      Sum Sq   Mean Sq F value    Pr(>F)
# 医院级别      3 6.8357e+13 2.2786e+13  3695.1 < 2.2e-16 ***
# Residuals 19963 1.2310e+14 6.1664e+09
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
4 mod.cor <- lm(总收入 ~ 诊疗人次 + 医院级别, data = hospital)
5 anova(mod.cor)
# Analysis of Variance Table
#
# Response: 总收入
#           Df      Sum Sq   Mean Sq F value    Pr(>F)
# 诊疗人次      1 1.3916e+14 1.3916e+14 57147.30 < 2.2e-16 ***
# 医院级别      3 4.4239e+12 1.4746e+12   605.58 < 2.2e-16 ***
# Residuals 19487 4.7453e+13 2.4351e+09
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

根据上面的解释，进行协方差分析肯定要包含回归分析和方差分析两部分。上述代码中，第1

行代码筛选出医院中二级和三级医院；第 2 行代码根据因子变量使用 `lm` 函数构建一个回归一元一次线性模型，回归模型应该你将要见到的模型中最简单的模型了，第 1 个参数告诉我们要构建什么样的表达式；第 3 行代码使用 `anova` 函数进行协方差分析，发现医院级别对目标变量的影响极显著；第 4 行代码添加不怎么可控的诊疗人次再次构建线性模型；第 5 行代码进行协方差分析发现， $\text{Pr(>F)}$  均小于 0.01，说明诊疗人次和医院级别均对医院收入产生了极显著的影响，另外，均方残差由  $6.1664\text{e}+09$  降到了  $2.4351\text{e}+09$ 。有了这个模型，我们就可以回答如果诊疗人次相同时（很难控制诊疗人次，因为药不能因为实验停），不同级别之间的医院的收入存在多大的差距。

- 调整后均值比较

```
1 aggregate(总收入 ~ 医院级别, data = hospital, FUN = mean)
#   医院级别    总收入
# 1    二级  33725.47
# 2    三级 255158.99
2 data.predict <- data.frame(医院级别 = factor(c("二级", "三级")), 诊疗人次 =
rep(183239, 2))
3 data.predict
4 adjmeans <- predict(mod.cor, data.predict)
5 adjmeans
6 data.frame(data.predict, adjmean = adjmeans)
#   医院级别 诊疗人次  adjmean
# 1    二级   183239  57584.56
# 2    三级   183239 122956.11
```

上述代码中，第 1 行代码比较计算在不去除诊疗人次影响的情况下，二级和三级医院的收入均值分别为 3373 万元和 25515 万元，二者的差距将近 2 亿元；第 2 行代码构建一个可以输入模型的数据框，数据框必须包含模型所需的所有自变量，且数据类型保持一致，保持诊疗人次不变为诊疗人次的平均值；第 4 行代码使用 `predict` 函数预测医院的总收入，也就是说如果医院级别和诊疗人次与 `data.predict` 一样的医院，那么总收入是多少，在使用 `predict` 函数预测时，第 1 个参数用于指定已经构建完成的模型，这里我们指定了上一步构建的模型 `mod.cor`，第 2 个参数指定输入模型数据，数据一定要和建模时的结构一样，一定要包含所有的输入变量（自变量），可以看到 `data.predict` 包括两列：医院级别和诊疗人次，而且数据类型也和模型构建时的数据相同；第 5 行代码查看预测结果；第 6 行代码将预测结果和 `data.predict` 捆在一起；我们看到当诊疗人次保持为 183239 时，二级医院和三级医院的收入均值变成了 5758 万元和 12295 万元，二者的差距变成了 1 亿元以内，这就是协方差分析的其中一个应用。

另外，在进行协方差分析时也需要符合几个重要的前提假设，首先包括正态性、方差齐性等，还包括因变量和协方差之间的直线关系及组内回归系数同质性等条件，这里不再展开。

为什么前两章的图表比较粗糙？因为这里的图表都和统计相关，需要一定的知识背景，并非是对普通大众的可视化，换句话说是不能见人的，另外，前两章属于入门阶段，重点在数据的整理和 R 语言的基本知识，不宜过多地讲解可视化内容，我们将在第 3 章详细讲解 R 语言中的可视化，也将对可视化的要求提高到专业的角度。



### 2.4.4 很多熟悉的数据处理方法已经成笑话，工具箱该换了

经过这两章的“死磕”，大家对 R 语言及数据处理应该比较熟练了，一般的数据统计分析工作应该可以顺利开展，这里我们主要对这两章中数据处理的操作做一次总结和提升，因为面对超大数据时（我实在不愿意使用大数据这个词，感觉它已经被滥用到无以复加的程度）能够提高速度。从接触 R，我们就已经开始熟悉各种问题的解决办法，并把它们记在心里。Stack Overflow 上有很多问题的答案，我们这里对一些高票答案进行了整理，因为这类高票答案往往在速度上和简洁性上超过一般方法，所以现在知道这些高效的答案并不是坏事。有时候作为技术人员要时刻准备更新自己的观念、知识和技能，这是技术员的宿命。如果一个技术员失去了开放的大脑，那么很容易就能从他的身上嗅出“大叔”的味道，因为他已经老了，更严重的是，在职业发展的道路上要不被技术淘汰，要不转做管理人员。是的，技术就是这样，一夜之间你以前所熟悉的答案，已经成了现在的笑话。

#### 1. 数据框按照一列或多列排序

根据变量（列） $z$  和  $b$  排序， $z$  按照降序， $b$  按照升序。首先创建一个足够大的数据集，然后分别使用不同的方法看一下效果，如下所示。

##### • 数据集

```
1 require(plyr)
2 require(data.table)
3 require(dplyr)
4 require(taRifx)
5 require(doBy)
6 require(taRifx)
7 require(dplyr)
8 set.seed(201635)
9 dat <- data.frame(b = as.factor(sample(c("Hi", "Med", "Low"), 10000000,
TRUE)), x = sample(c("A", "D", "C"), 10000000, TRUE), y = sample(100, 10000000,
TRUE), z = sample(5, 10000000, TRUE), stringsAsFactors = FALSE)
```

排序经常被国际比赛用来测试系统的速度，我们加载了很多包，它们多数都提供排序的方法，然后生成一个包含 4 个变量 1000 万行的数据框，作为排序的备用数据，大家看到我在生成数据框时指定了随机种子。为了加快写作过程，我这里仅测试了 1000 万行数据，你有时间也可以测试更多，但是小心笔记本电脑崩溃。另外，同样为了节省时间，我们这里的比较不再进行统计意义上的比较，有时间你可以按照上面的均值比较的内容进行方法之间速度和内存占用的比较，推荐使用 benchmark 包。生成数据时主要使用了 sample 函数，按照放回式抽样抽取 1000 万次。

##### • 排序

```
1 system.time(temp <- dat[with(dat, order(-z, b)), ]) ## 来自 R 基础包
# 用户 系统 流逝
# 22.94 0.10 23.20
2 system.time(temp <- orderBy( ~ -z + b, data = dat)) ## 来自 doBy 包
# 用户 系统 流逝
```

```
# 32.74 0.16 33.20
3 system.time(temp <- plyr::arrange(dat, desc(z), b)) ## 来自 plyr 包
# 用户 系统 流逝
# 22.23 0.06 22.32
4 system.time(temp <- dplyr::arrange(dat, desc(z), b)) ## 来自 dplyr 包
# 用户 系统 流逝
# 7.41 0.01 7.86
5 system.time(temp <- sort(dat, f = ~ -z + b)) ## 来自 taRifx 包
# 用户 系统 流逝
# 33.06 0.09 33.39
6 setDT(dat)
7 system.time(temp <- dat[order(-z, b)])
# 用户 系统 流逝
# 0.59 0.02 0.69
8 system.time(temp <- setorder(dat, -z, b))
# 用户 系统 流逝
# 0.51 0.01 0.53
```

上述代码中，第 1 行代码使用了 `system.time` 函数，第 1 章我们使用了 `system.time` 函数，它用于返回代码执行所用的时间，`order` 函数用于对数据排序，返回的是原数据的行编号，第 1 章我们已经讲过，这里使用了 `with` 函数，它用于指定 R 程序所用的数据源，比如在不使用 `with` 函数的情况下，我们需要这么写：`dat[order(-dat$z, dat$b), ]`，而用 `with` 函数指定数据源为 `dat`，就不用使用“\$”符号提取指定了，`order` 函数前面已经讲过了，在列名称添加减号表示按降序排列，可以看到 R 基础包的排序耗费了 23 秒；第 2 行代码中 `doBy` 包的 `orderBy` 函数也能完成排序的效果，它的第 1 个参数是一个表达式，表示数据框按照 `z` 降序 `b` 升序排列，共耗时 33 秒；第 3 行代码 `plyr` 和 `dplyr` 包都有 `arrange` 函数用于排序，如果两个包中存在重名函数，先加载的包的函数就会被后加载的包的函数替换掉，所以为了避免被替换，在使用函数时需要注明使用哪个包里的，即在函数前面添加包名称，中间用两个英文冒号隔开，如 `plyr::arrange` 表示使用 `plyr` 包里的 `arrange` 函数，它的第 1 个参数为数据框，后面的参数用于指定按照哪些列排序，可以添加任意多个，如果降序需要在列名称前面使用一个 `desc` 函数，表示按其降序排列，`plyr` 包使用了 22 秒；第 4 行代码中 `dplyr` 包的 `arrange` 函数用了 7.86 秒，速度有提升；第 5 行代码中 `taRifx` 包的 `sort` 函数也可用于排序，参数与 `orderBy` 函数比较相似，第 1 个参数指定数据，第 2 个参数指定一个表达式，速度比较差；`data.table` 包是最近流行的一个处理数据速度超快的包，但是我对它是否存在 bug 表示怀疑，使用时一定要检查结果是否按照操作要求执行，检查结果的方法可以使用 `tail`、`head`、`sample` 函数；而第 6 行代码的 `setDT` 将数据转化为 `data.table` 对象，并设置了索引；第 7 行代码重新对 `dat`（此时已经是 `data.table` 对象）排序，仍然使用基础函数 `order`，耗时降到 1 秒以内；如果我们使用 `data.table` 自带的 `setorder` 函数进行排序呢？方法很简单，只需要指定数据和按哪些列重排就可以了，速度又降低了，我曾经测试过，1000 万行数据最好成绩为 0.08 秒；`data.table` 的排序速度非常快，因为它使用了基数树（`radix ordering`）。`DT[order(...)]`虽然是 R 原有的语法，但是已经经过了 `data.table` 对象方法的优化，速度非常快，内存

占用也比较少。data.table 中的方法不仅提高了速度，而且降低了对对象占用内存的量，你可以使用 object.size 函数查看 dat 为数据框时和为 data.table 对象时内存占用的大小，setorder 函数甚至能减少排序过程中内存占用的最大峰值。

## 2. 比较\*pply 家族、by 及 aggregate 透视表函数

前面已经详细介绍过\*pply 家族函数，其实它们是 R 用来快速处理循环过程的函数，只是它们的输入和产出略有不同，也就造成了它们名字的首字母不同。但是我们今天从另外一个角度比较它们，即它们的效率问题。其实不同的函数所要求的输入和输出对象不同，但是本质上它们都是对某一对象的分组和分步计算，apply 函数要求的输入与其他函数差异较大造成调整时间比较大，这里就不把 apply 函数列入讨论范围了。

我们要完成将 1000 万行数据按照 5 万个因子分组求和、计数，就是 sum 和 length 函数，我们测试一下\*pply 家族和 data.table、dplyr 包提供的函数完成上述测试需要的时间，你可以把表现较好的函数加入自己的工具箱了，但是无论什么时候都要保持对机器的怀疑，一定要记得检查结果。

- 创建测试数据

```
1 set.seed(201635)
2 n <- 10000000
3 k <- 50000
4 x <- runif(n)
5 hist(x)
6 grpfactor <- sample(k, n, TRUE)
7 length(x)
8 length(grpfactor)
```

上述代码中，第 1 行代码设置了随机种子；第 2 行代码创建对象 n，等于 1000 万行数据；第 3 行代码创建对象 k 等于 5 万行数据；第 4 行代码使用 runif 函数按照均匀分布抽取 1000 万个数值；第 5 行代码使用 hist 函数绘制频率直方图，不难看出 x 的值分布在 0~1 之间，每个区间的个数基本相等，这就是均匀分布的特征；第 6 行代码抽取 5 万个分组数据，sample 表示从 1:k 的序列按照放回的方式抽取 n 个数值，然后赋值给 grpfactor；最后两行代码查看两个对象的长度，发现它们都包含 1000 万个值。

- 测试 sapply 的速度

```
1 system.time({
2   lt <- split(x, grpfactor)
3   res.sapply <- sapply(lt, function(x) list(sum(x), length(x)), simplify
= T)
4   res.sapply <- t(res.sapply)
5 })
# 用户 系统 流逝
# 2.27 0.05 2.31
6 head(lt)
7 tail(res.sapply)
```

`system.time` 函数如果要获得多行代码的速度，则需要在代码块外面添加一个花括号，表示这是一个完整的代码块，比如我们在两行代码的外面使用了花括号，第 1 行代码我们使用 `split` 函数将 `x` 按照 `grpfactor` 分组，其结果是一个 `list`，每个元素为被分为同一个小组的 `x` 值；第 3 行代码将这个 `list` 的每一个元素使用 `sapply` 函数传递给匿名函数，匿名函数计算每个小组的数据之和与长度，`simplify = T` 表示将结果转为矩阵，然后将结果转置一下；最后两行代码检查数据，看到程序耗时 2.48 秒，`split` 函数速度非常慢，建议大家不要设计使用 `split` 函数的程序。

- 测试 `lapply` 的速度

```
1 system.time({
2   lt <- split(x, grpfactor)
3   res.lapply <- lapply(lt, function(x) list(sum(x), length(x)))
4   res.lapply <- do.call(rbind, res.lapply)
5 })
# 用户 系统 流逝
# 2.31 0.03 2.34
```

之前讲过 `sapply` 和 `lapply` 其实是一样的，`sapply` 就是添加了 `simplify` 参数的 `lapply`，所以它们的速度比较相似，`lapply` 生成的是一个 `list`，需要使用 `do.call` 函数调用 `rbind` 函数将它的元素按行捆绑成矩阵，耗时没有改观。

- 测试 `by` 的速度

```
1 system.time({
2   res.by <- by(x, list(grpfactor), function(x) list(sum(x), length(x)))
3   res.by <- do.call(rbind, res.by)
4 })
# 用户 系统 流逝
# 6.24 0.16 6.39
```

`by` 函数与 `*pply` 家族也有着千丝万缕的关系，它是 `tapply` 的变形，函数的第 1 个参数指定被分组的对象，第 2 个参数指定用于分组的 `list`，所以这里将向量 `grpfactor` 转化为了 `list`，然后将分组结果传递给匿名函数，函数耗时较多。

- 测试 `aggregate` 的速度

```
1 system.time(
2   res.aggregate <- aggregate(x, list(grpfactor), function(x) list(sum(x),
length(x)), simplify = FALSE)
3 )
# 用户 系统 流逝
# 20.46 0.55 21.34
```

前面我们讲过使用 `aggregate` 函数制作透视表的操作，使用的对象为数据框，但是对象为向量时它的用法和 `by` 函数一样，先分组然后按照分组结果传递给匿名函数，返回的结果仍然是一个 `list`，耗时超过了 20 秒。

- 测试 `dplyr` 包的透视表速度

```
1 library(dplyr)
```

```

2 system.time({
3   df <- data_frame(x, grpfactor)
4   grp <- group_by(df, grpfactor)
5   res.dplyr <- summarise(grp, sum(x), n())
6 })
# 用户 系统 流逝
# 1.92 0.01 1.93

```

使用 dplyr 包做透视表前面也已经讲过，上面代码先将数据合并为数据框，这里使用的 data\_frame 函数来自 dplyr，其作用和基础包的 data.frame 一样；group\_by 函数将数据框按照 grpfactor 分组，summarise 函数将分组结果传递给调用的函数，这里调用了 sum 函数和 n 函数，用计数函数 n 替换了 length 函数，dplyr 包比上面的其他方法好像有点速度优势。

- 测试 data.table 包的透视表速度

```

1 library(data.table)
2 system.time({
3   dt <- setDT(list(x, grpfactor))
4   dt <- setnames(dt, c("x", "grp"))
5   res.data.table <- dt[, .(sum(x), .N), grp]
6 })
# 用户 系统 流逝
# 0.47 0.01 0.53

```

data.table 包也能做透视表，setDT 函数将 list 转化为 data.table 对象，上面已经说过它还能将数据框转化为 data.table 对象，setnames 函数用于重新设定 data.table 对象的列名称，DT[i, j, by] 中 i 表示被分组的对象，by 表示分组列，j 表示被用于的计算过程，一般为函数，dt[, .(sum(x), .N), grp] 对 data.table 对象的每一行按照 grp 分组之后传递给函数组合 sum 和 N，分别求和、计数，这里的 dot 表示执行的意思。

### R 中不同 dot 的意思

- (1) 在对象名称中只是个字符，如 res.dplyr。
- (2) 在函数的参数中，表示省略参数，如 rbind(..., deparse.level = 1)。
- (3) 表示继承关系，如我们绘制线性模型 plot(lm(x ~ y)) 的时候，plot 函数会搜寻 plot.lm 函数，这里就表示继承关系。
- (4) 表示省略变量名称，如 lm(species ~ ., data = iris)，表示使用 iris 中除 species 之外的所有变量作为自变量。

(5) .() 表示执行括号里的语句或函数，可以输入 “?bquote” 查看用例。

我们看到 data.table 的耗时已经降到 1 秒以内。其实我觉得这种方法测试并没有达到真正的公平公正，因为先测试的方法得出的结果保存在内存中，由于数据量比较大，R 对象占用的内存会越来越多，肯定会影响后面的方法的效率，但是从结果上看，data.table 仍然当仁不让地成为最高效的方法，可怜我一直喜爱的透视表 aggregate 函数效率太低，姑且假装它受了前面那些“坏蛋”的影响吧。

### 3. 怎么极速关联（合并）两张表

我们曾介绍过一些用于表关联的函数和方法，但是没有比较过它们的速度，这里做个总结测试一下它们的效率，如下所示。

- 不同方法取交集

```
1 rm(list = ls())
2 library(sqldf)
3 library(dplyr)
4 library(data.table)
5 n <- 10000000
6 set.seed(123)
7 df1 <- data.frame(x = sample(n, n), y1 = rnorm(n))
8 df2 <- data.frame(x = sample(n, n), y2 = rnorm(n))
9 dt1 <- as.data.table(df1)
10 dt2 <- as.data.table(df2)
11 system.time(temp <- merge(df1, df2, by = "x"))
# 用户 系统 流逝
# 53.72 2.13 105.24
12 system.time(temp <- sqldf("SELECT * FROM df1 INNER JOIN df2 ON df1.x = df2.x"))
# 用户 用户 系统 流逝
# 105.24 235.16 350.49
13 system.time(temp <- inner_join(df1, df2, by = "x"))
# 用户 系统 流逝
# 12.97 0.36 16.75
14 system.time(temp <- dt1[dt2, nomatch = 0L, on = "x"])
# 用户 系统 流逝
# 3.26 0.04 3.31
```

上述代码中，第 1 行代码清除内存中的对象，以免影响测试速度；第 2~4 行代码加载相应的包；第 5 行代码创建一个对象 n；第 6 行代码设置随机种子；第 7 行代码创建一个包含两个变量 1000 万行数据的数据框，这里的 sample 函数抽样时我们选择了非放回式抽样；第 8 行代码创建另外一个数据框；第 9~10 行代码将数据框转化为 data.table 对象；第 11 行代码使用 merge 函数取两个表的交集，耗时 105 秒；第 12 行代码的 sqldf 包能让你像在数据库上一样写 SQL 来操作数据表，但是其关联速度要远比在数据库上慢，耗时 350 秒；而第 13 行代码 dplyr 包的 inner\_join 函数仅用了 16.75 秒；如果将数据框转化为 data.table 对象再进行关联速度会有不错的提升，第 14 行代码 nomatch = 0L 参数表示取交集 on 参数指定所用的 id 列。

既然 data.table 包的效率那么高，我们就有必要讲一讲它进行各种关联关系时的操作了，如下所示。

- 使用 data.table 包进行表关联

```
1 setkey(dt1, x)
2 setkey(dt2, x)
# 右连接
```

```

# 相当于 dplyr::left_join(df2,df1)
3 system.time(temp <- dt1[dt2, on = "x"])
# 左连接
# 相当于 dplyr::left_join(df1, df2)
4 system.time(temp <-dt2[dt1, on = "x"])
#取交集 (inner join)
5 system.time(temp <- dt1[dt2, nomatch = 0L, on = "x"])
#取匹配不到的集合
6 system.time(temp <- dt1[!dt2, on = "x"])
#merge 函数取交集 (inner join)
7 system.time(temp <- merge(dt1, dt2, by = "x"))
# 用户 系统 流逝
# 0.67 0.09 0.81

```

上述代码中，第1行和第2行代码使用 `setkey` 函数给 `data.table` 对象构建索引，第1个参数指定 `data.table` 对象，第2个参数指定将哪一列设置为索引；第3行代码进行右连接，即保证 `dt2` 的数据都在，将 `dt1` 的数据添加到 `dt2`，相当于 `dplyr::left_join(df2,df1)`，我们看到添加了索引的关联比上一步没有添加索引的关联更加快；第4行代码进行左连接；第5行代码取交集；第6行代码取匹配不到的数据；第7行代码表示经过 `data.table` 改造之后的对象再使用基础包的 `merge` 函数，速度同样有提升。

#### 4. R 数据读取速度的比较

很多人抱怨 R 读取数据比较慢，这里我们对比 R 中几种读取数据的方式，看看哪个速度快，从中挑选一个加入工具箱。

- 生成测试数据

```

1 library(data.table)
2 n <- 10000000
3 df <- data.frame(a = sample(1:1000, n, replace = TRUE), b = sample(1:1000,
n, replace = TRUE), c = rnorm(n), d = sample(c("foo", "bar", "baz", "qux", "quux"),
n, replace = TRUE), e = rnorm(n), f = sample(1:1000, n, replace = TRUE) )
4 write.csv(df, "test.csv", row.names=FALSE)

```

上述代码会生成一个比较大的数据表，第1行代码加载了 `data.table` 包；第2行代码创建一个对象 `n`；第3行代码创建一个数据框 `df`，它包含6列1000万行数据；第4行代码将这个表写到当前目录下的 `test.csv` 文件，这个对象的大小约390MB，不算很大。

- 快速读取数据

```

1 rm(list = ls())
2 gc()
3 system.time(Df1 <- read.csv("test.csv", header = T, sep = ",", stringsAsFactors
=FALSE))
# 用户 系统 流逝
# 94.69 1.07 135.30

```

```
4 rm(list = ls())
5 gc()
6 system.time(DT <- fread("test.csv", sep = ",", header = T, stringsAsFactors=FALSE))
# Read 10000000 rows and 6 (of 6) columns from 0.381 GB file in 00:00:27
# 用户 系统 流逝
# 22.03 0.17 26.82
7 rm(list = ls())
8 gc()
9 require(sqldf)
10 system.time(SQLDF <- read.csv.sql("test.csv", dbname=NULL))
# 用户 系统 流逝
# 45.96 0.70 47.41
```

上述代码中，第 1 行代码删除内存中的 R 对象；R 里面即使对象被移除后，内存也不会马上被回收，需要使用 gc 函数回收内存，这样下面的测试压力就比较小一点（见第 2 行代码）；第 3 行代码使用 read.csv 函数读取数据，耗时 135 秒；第 4 行和第 5 行代码移除内存中的对象回收内存；第 6 行代码使用 data.table 包的 fread 函数读取数据，耗时 27 秒；第 7 行和第 8 行代码行筛出对象回收内存；第 9 行代码加载 sqldf 包；第 10 行代码使用 read.csv.sql 函数读取数据，耗时 47 秒。

删除一个数据框中的一列或多列，或者按存储在某个变量中的列名称批量删除，data.table 包中设定的函数同样可以完成，既然 data.table 包那么有优势，学习点基本操作还是没错的。下面分别介绍。

- 使用 data.table 包移除列

```
1 DT <- fread("test.csv", sep = ",", header = T, stringsAsFactors=FALSE)
2 DT[, c('a','b') := NULL]
```

需要提及的是，在删除列时，大家不要使用编号索引删除，例如“df[, -c(1,4)]”这种形式不建议在代码中出现，因为别人看不懂你要删什么，特别是 df 的格式一旦发生变化后，恐怕你自己也很难分清当时 1 和 4 列到底指的是什么，所以请尽量使用列名删除。同样，当你筛选或提取时也要尽量使用列变量的名称，而不是它们的索引编号，上述代码的第 2 行表示将 a,b 两列的值赋值为 NULL，NULL 在 R 中表示空，空就是没有，缺失表示有，但是缺失了，用 NA 表示。

在数据整理阶段，通常要修改一下数据集里的列名，除了常见的修改方法外，data.table 包的 setnames 函数也能用于修改数据框中的列名，如下所示。

- 使用 data.table 包重命名列

```
1 rm(list = ls())
2 gc()
3 library(data.table)
4 set.seed(123)
5 n <- 1e7
6 dt <- data.table(a = sample(1:3, n, TRUE), b = rnorm(n))
7 address(dt)
# [1] "0000000017F76860"
```



```
8 setnames(dt, c("m", "n"))
9 address(dt)
# [1] "0000000017F76860"
```

setnames 采用索引的方式修改，新加占用内存要比常见方法少，可以通过 address 函数看看对象的内存指针地址，如果内存指针地址发生改变，则说明上步操作有可能复制了对象，如果没有改变就是在原有对象上的修改。如果你的内存比较紧俏，还是选用 data.table 提供的方法比较合适。

## 5. 将 list 转化为数据框

将一个 list 的不同元素包含的内容按行组合为一个新的数据框。其实，完成数据对象之间的转换是数据分析师的基本技能，除了将 list 转化为 data.frame 外，你还要学会将 data.frame 转化为 list、list 转化为 corpus、因子转化为字符等，前面的章节已经分别讲过了，我们现在注意的是要完成这个任务采用不同方法是否存在效率上的差异。

我一般使用基础的 do.call 和 rbind 函数，如果 list 各个元素的长短不一，也要选择不同的处理方法，参看第 11.4 节。但是 data.table 里的 rbindlist 不仅解决了长短不一（使用 fill 填充）的问题，而且还可以根据 list 中元素的名称匹配后再按行粘贴，当然这些方法如果都设置使用，则会影响其效率，但无论如何均比基础包里的 do.call 和 rbind 函数组合的效率要高。

- 构建数据集

```
1 rm(list = ls())
2 gc()
3 DF1 <- read.csv("test.csv", header = T, sep = ",", stringsAsFactors=FALSE)
4 lsbig <- split(DF1, DF1$F)
5 system.time(res1 <- do.call("rbind", lsbig))
# 用户 系统 流逝
# 360.71 49.28 512.84
6 system.time({
7   res2 <- rbindlist(lsbig, use.names=TRUE)
8   res2 <- as.data.frame(res2)
9 })
# 用户 系统 流逝
# 0.41 0.32 29.55
```

上述代码中，第 1 行和第 2 行代码删除对象回收内存；第 3 行代码读取 2.4.3 节制造的大数据框；第 4 行代码使用 split 函数将其 DF1 按照 DF1\$F 分组为 list，这个 list 包含 1000 个元素，每个元素是一个 10000×6 的数据框；第 5 行代码使用 do.call 调用 rbind 的方法将其转化为数据框，前面的章节讲过，耗时 513 秒；第 6 行代码使用 data.table 包的 rbindlist 函数将 list 转化为数据框，其结果是 data.table 对象，使用 as.data.frame 将结果转化为数据框，共耗时半分钟不到。

当你的方法让机器人遍体鳞伤时，主动去重新发现也许能够暂解燃眉之急，所谓精通就是方法的再发现。

## 6. 更加高效的长宽表转换方式

之前讲过最常见的数据变形为长宽表的变换，刚好 reshape2 包和 data.table 包都有同名函数，我们这里不妨测试一下它们的效率，以方便在碰到极端情况时有一个备选工具。

- 生成测试数据

```
1 n <- 1000000
2 k <- paste(rep("var", 30000), 1:30000, sep = "")
3 df <- data.frame(id = sample(1:1000, n, replace = TRUE), var = sample(k,
n, replace = TRUE), value = rnorm(n))
```

上述代码中我们生成了一个长型表，包括 3 个变量 500 万条数据，首先我们要把它变成宽表，然后再变换回来，分别测一下 reshape2 包和 data.table 包的速度，如下所示。

- 使用 dcast 函数操作

```
1 system.time(res.reshape <- reshape2::dcast(df, id ~ var, value.var =
"value", fun.aggregate = sum))
#   用户   系统   流逝
# 104.24 11.25 1218.72
2 dt <- data.table::data.table(df)
3 system.time(res.data.table <- data.table::dcast(dt, id ~ var, value.var
= "value", fun.aggregate = sum))
# 用户 系统 流逝
# 2.73 0.11 8.75
4 head(res.data.table[,.(id, var1)])
5 temp <- data.frame(res.data.table)
```

上述代码中，第 1 行代码我们使用了 reshape2 包的 dcast 函数，如果你不想加载某个包，只想使用其中的某个函数，就在包名和函数名之间添加 "::"，这样就只临时加载了函数，测试发现 reshape2 包用了 1218 秒；第 2 行代码将数据框转化为 data.table 对象；第 3 行代码使用 data.table 包的 dcast 函数，用法与 reshape2 包一样，耗时变为 8.75 秒；第 4 行代码你可以查看前两列的内容；第 5 行代码也可以将 data.table 转化为数据框。

- 使用 melt 函数操作

```
1 system.time(df <- reshape2::melt(res.reshape, id.vars = c("id"), variable.name
= "var", value.name = "value"))
#   用户   系统   流逝
#   1.11   1.84 60.38
2 df <- df[!df$value == 0,]
3 system.time(dt <- data.table::melt(res.data.table, id.vars = c("id"),
variable.name = "var", value.name = "value"))
#   用户   系统   流逝
#   0.33   0.18 13.33
4 dt <- dt[!dt$value == 0,]
```

上述代码中，第 1 行代码使用 reshape2 包的 melt 函数将上一步的结果重新融合为长表；融合之后会产生很多 value 等于 0 的行，这些行是不需要的，第 2 行代码通过筛选的方式将其去除，去除之

后还剩不到 100 万行；第 3 行代码使用 `data.table` 包将上一步的结果重新融合为长表，其应用方式和参数与之前讲过的几乎一样；第 4 行代码删除 `value` 等于 0 的行。

其实 `dcast` 函数面对超大数据时还是显得苍白无力的，至少会出现内存不够用的情况，这时你就需要理解自己的分析过程，将长表变宽表的过程分拆为多个步骤执行，我们将在第 14 章详细讲解。

除了上面的方法以外，`data.table` 中的“`%chin%`”匹配函数要比基础的“`%in%`”快一些。通过上面的例子大家应该比较深刻地感受到 `data.table` 提供的解决方案带来的效率上的快速提升。其实我想说的是在数据分析、数据挖掘的过程中，不断地发现新方法创造新效率才是进步的阶梯。不管是你多么熟悉多么膜拜的方法，如果它让你在某些工作面前捉襟见肘、手足无措，就应该勇于替换掉它们，寻求新的技能。

我也看过不少关于 R 的书，个人比较“崇洋媚外”，国内的好书还是比较少的，多是些应景之作。曾经有人建议我将本书按章节分为入门、进阶和精通几个篇章，我个人不喜欢这种方法，实际上很多精通和进阶的内容都是对入门时基础操作的颠覆，而不是上面基于项目进程中的分法。即使在项目进程中，基础的数据清洗和整形的重要性、难度、耗时都比后面的算法选择之类的操作要高出很多，所以在本书中将精通定义为解决方案的重新发现。

另外，通过本节你可能已经发现 R 中的函数多如牛毛，怎么办？其实学习高级语言，不能停留在熟记函数的级别，而是知道这个语言的优缺点和最基本的语法操作，至于函数，只要有了问题，我们都可以成为代码的搬运工，Google、百度都可以帮到你，找到函数之后查看函数的帮助文档能够帮助你迅速使用函数，何必劳心劳力地记忆根本记不完的东西呢？

## 第 3 章

# 从商务气质的数据可视化说起

### 3.1 说说数据可视化的专业素养

数据可视化不是简单的绘图，从原来的平面图表到现在的 D3.js 的互动图表，已经超越了原有的图表绘制的概念。但是万变不离其宗，有很多专业的知识和素养还是需要初学者逐步建立并在以后的职业生涯中持之以恒地践行的。下面一起探讨一下在如今浩如烟海的可视化作品中，我们应该具备哪些始终如一的素养和专业技能。

#### 3.1.1 数据可视化历史上有多少背影等你仰望

为什么要在这里讲述数据可视化的历史，原因很简单，诗经小雅有云：“他山之石，可以攻玉”，知道数据可视化的各个历史阶段，能迅速定位自己的图表处于何种阶段，鞭策提高自己。现代数据可视化早已不是仅将数据映射为图形的原始概念，它已经成为科学、统计、艺术的综合体。这里的艺术对现代图形设计者很重要，换句话说，现代数据可视化不仅体现了你的科学素养和敏锐的眼光，还标志着你的格调。现代人一天刷微博浏览的奇人轶事就是蒲松龄《聊斋志异》的信息量的好几倍，因此这里讲一下数据挖掘的历史，方便大家超越前者。

中国、古巴比伦、古罗马等文明古国在很久之前就已经技艺娴熟地使用图表记录星相移动、航海地图、季节变化、城市布局等方面的数据信息，如果非得要给数据可视化追溯一个历史起源，那么结绳记事就应该是数据可视化的源头了，大事打大结，小事系小结，不就是现代的气泡图吗，只不过缺少  $Y$  轴罢了（ $X$  轴为时间序列）。纸张发明之前（不要问我纸张发明之前人类怎么上厕所），人类就将信息图雕刻在岩壁上，纸张这个古老的载体发明之后人们就将信息写在纸上更加容易分享传播，现在信息的载体成了你的硬盘。

地图应该是现代图形的起源，因为土地是几千年农耕文明的生产资料，无论是国家还是农奴，估计最想知道的是这个世界有多大？重要的是地图对于军事和贸易不可或缺，因此古罗马成了最著名的绘制地图的帝国。公元前 366—335 年绘制在羊皮纸上的波伊廷格古地图覆盖了西起大不列颠东

至恒河的罗马帝国的完整交通路网，共标注了 2769 个村镇的位置及道路距离，无论是研究地图还是星相都是为了征服管理更多的领土，更好地进行贸易往来。

除了生理进化以外，交易是人类社会创新的根本动力。——《理性乐观派》

公元 950 年，欧洲出现了一幅具有现代图形意义的作品，网格系统第一次出现，也出现了数轴的雏形（时间和天体），它描绘了天体位置随时间的变化，但更加详细的信息现在已经很难解读了。

别急，咱们中国人也没闲着，别以为中国的精英都考科举去了，任何一个文明的国家在以交易为主角的市场中如果没有创新，都不可能保持统治地位。北宋时期除了沈括，还有一个非著名的苏颂，他头上有好几个“家”的荣耀，诸子百家、“三教九流”可谓是无所不通、无所不览，精通算法、地理、本草、音律等古代专业，另外，与沈括不同的是，他官拜宰相，死后追封魏国公。图 3-1 中最下面的图就是苏颂先生绘制的天体分布图，X 轴是已经将天空维度化的坐标，虽然比图 3-1 上面图所示欧洲的图形晚 100 多年，但是图中所用技术直到 16 世纪才传入欧洲，在中国可视化历史上具有重要的里程碑意义。

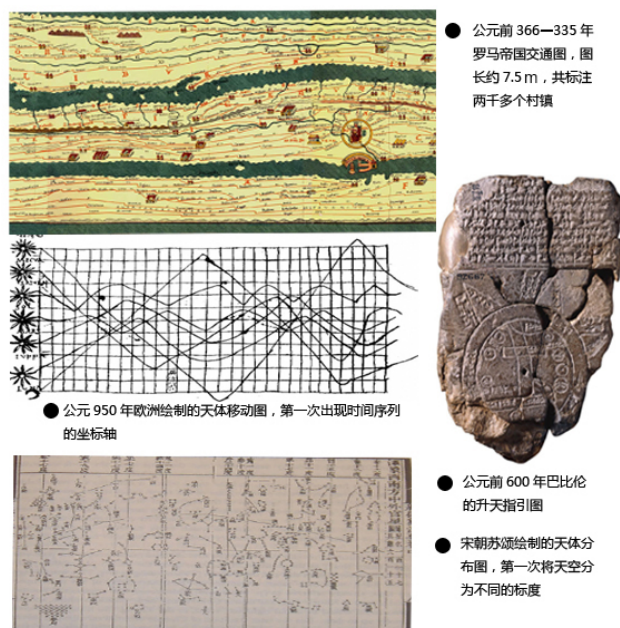


图 3-1

公元 1300—1800 年是数据可视化图形基础元素及理论创造和发生的年代，出人意料的是，最早出现的竟然是柱形图，但那时的柱形图和现在的柱形图意义不一样，比较像现代的折线图，用于描述一个变量随另外一个变量的变化过程。同时还先后出现了折线图、更加完备的世界地图、等高线图，以及折线图和柱形图结合的综合图形。

公元 1800—1899 年这一时期是数据可视化历史上的辉煌时期，几乎所有的现代统计图形均在这

一时期出现（见图 3-2）。1801 年，William Playfair 绘制了第一张饼图，分析土耳其帝国、欧洲、非洲的统治优势，随后南丁格尔的玫瑰图出现，奠定了极坐标面积图的基础，另外三维图也在该时期出现，同时还加入了现代摄影技术，绘制了物体位移时间图，但可惜的是，随后 1900—1975 年可视化进入了应用时期。南丁格尔玫瑰图使用扇形面积表达比例变化（其缺点后面会继续讲解），将极坐标分为 12 等份，每一扇代表一个月，扇面积代表一个月的死亡人数，每个月的死亡原因又分为 3 种，分别用不同的颜色标注，其实它是面积堆积柱状图的扭曲。由于当时图表数据很难引起官方机构的热情，南丁格尔遂使用彩色绘制图表，同时对图表增加了变形，让英国官方认识到战争中的真正死神是卫生护理条件差，当时英国士兵伤员死亡率 42%，南丁格尔通过改善卫生条件，仅半年内就将伤员死亡率降低到 2.2%，她也有了“提灯女神”的美称。图表除了基础的表达数据的能力以外，还要具有亲和力和美的视觉享受，从这一点上讲，南丁格尔是第一位为数据可视化引入艺术元素的开拓者。

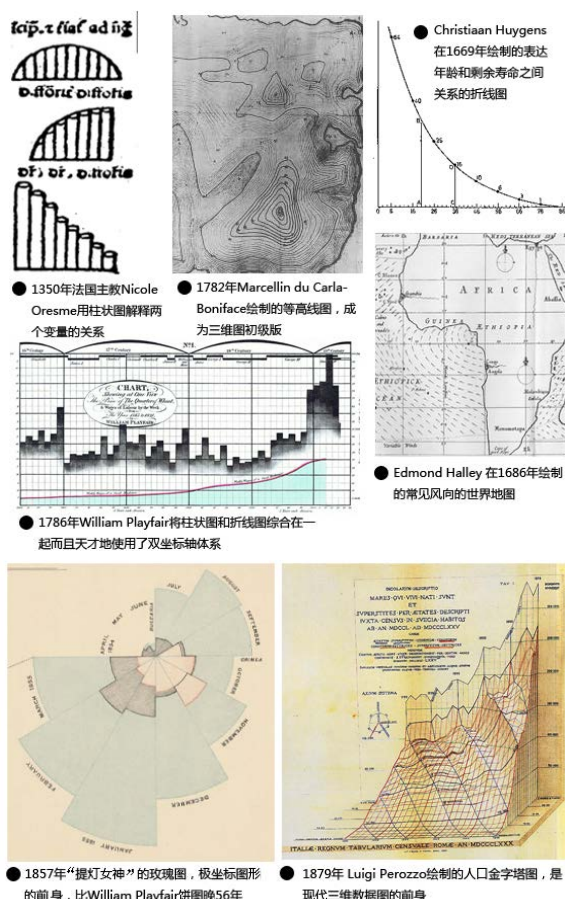


图 3-2

这一时期还出现了一幅数据可视化的巅峰之作，其历史背景是不可一世的拿破仑因为皇后未能生育，想要娶俄国沙皇亚历山大一世的妹妹为妻，但是亚历山大不同意，于是拿破仑就集结了 51 万人，包括 40 个步兵师和 25 个骑兵师入侵了俄国，途中每一战都历经艰险，最终到达莫斯科时又遭到俄国人坚壁清野策略的打击，之后无功而返。工程师 Charles Joseph Minard 非凡卓越地用一张图记录了这场灾难，此图分为上下两个部分，上半部分为进攻和撤退路线图，眼界卓绝地使用了数据流的方式展现了进攻和撤退时士兵人数的变化，上面浅灰色表示刚开始各个分支兵团沿途集结，条带很宽表示人数很多，随着战争的推进人数递减，行至 Dorogobouye 时赶上风雪，士兵忙着把尸体抬出营地，而下面黑色为撤退路线，从莫斯科撤退时还有 10 万人，渡过出发地涅曼河时还剩 10000 人，Minard 非常优秀地在一张二维图中展示了 6 种不同的数据类型：拿破仑军队的数量、行军距离、气温、经纬度、行军方向、特定日期的军队位置，这张图被誉为最优秀的统计图形，它也是现代统计图形 Sankey 的雏形。

1975 年以后至今，计算机作为先驱，数据可视化得以蓬勃发展，成为数据挖掘和数据分析行业与受众沟通的基本形式，数据可视化成了数据与受众沟通的语言，使数据发出决策的指令。本章的目的是领悟数据可视化的科学性、艺术性，数据可视化可以称得上在数据分析挖掘流程中唯一有着文艺气息的环节，所以要以专业主义的精神掌握。

一言以蔽之，数据可视化不仅将数据映射为图形，优秀的统计图形除了遵循基础的可视化原则以外，还要非常深刻地揭示数据背后所隐藏的规律和趋势，并以友好的方式引起受众的共鸣。统计图形本质上不是和数据结合，而是和数据反映的规律结合，所以这不是随意选择一种数据可视化图形的问题，数据可视化其实是一种设计，它和数据建模一样，是一个追求可视化和规律自洽的过程，以达到图形之前的受众和数据背后的规律心领神会、水乳交融的境界。因此，从阅读数据可视化的历史中我得到这样一点启示：数据可视化在基础图形上修修改改、综合叠加，甚至可以进行颠覆性设计，无须拘泥于基本图形，但这种水平我们身不能至，心向往之。

#### 3.1.2 商务图表应该具有哪些素质

什么叫专业，怎么做到专业？专业就是每一个细节都经得起推敲，做到专业首先要有一个态度：专业主义态度，这是由大前研一先生提出的，《经济学人》将大前研一评为“日本战略之父”，他认为专业的态度就是为了追求专业而付出努力。

我们想做到专业图表，方法有一个，就是模仿，模仿别人的每一个细节，先做到精神和态度上的专业，然后追求技能的专业，而商务图表最主要的一个气质就是专业。

##### 1. 图表与其表现的数据关系吻合

首先，商务图表的基础是要清楚图表表现的数据关系，专业要建立在关系找准的基础上。如果关系没找对，无论你的图表多么漂亮都将是徒劳的、无用的。图表要表现的关系可以概括为对比、趋势、相关等，找到要展示的数据关系然后才能选择合适的图表。

对比关系可以分为纵向对比和横向对比，我们将纵向对比定义为内对比，比如一个市场内三家

企业份额的对比即为纵向对比，横向对比为外对比，比如不同市场内的两家企业份额对比或者同一个市场的企业不同时段比较即为横向比较。

纵向对比最常见的就是分蛋糕关系，所有对比对象总和为一个整体，不符合这一原则的就只能用横向对比，需要提醒一下，并不是所有的百分比数据都可以使用纵向对比，纵向对比比较常见的图包括饼图、环图和条形堆积图等。

横向比较常见的图表包括柱形图、条形图和雷达图等，比如比较几个同学的成绩，比较不同网站访问类型的比例（百分比数据），所以并不是所有的比例数据都可以用饼图。如图 3-3 所示，右边是堆积图和堆栈图，它们都是可以同时进行横向对比又可以进行纵向对比的图，不同的习惯决定是用堆栈图还是堆积图好，比如右边第一张图我们习惯上会将普通电话和 400 电话看成一个整体，所以用堆积图，但是我们很少将本月和上月的销量看成一个整体，至少也要够一个季度才作为整体，所以这里比较适合用堆栈，右边上下两张图和关系的对应都是比较正确的，就是丑了点儿。如果上下两张图换一下，就会犯选择基础图形不专业的错误。

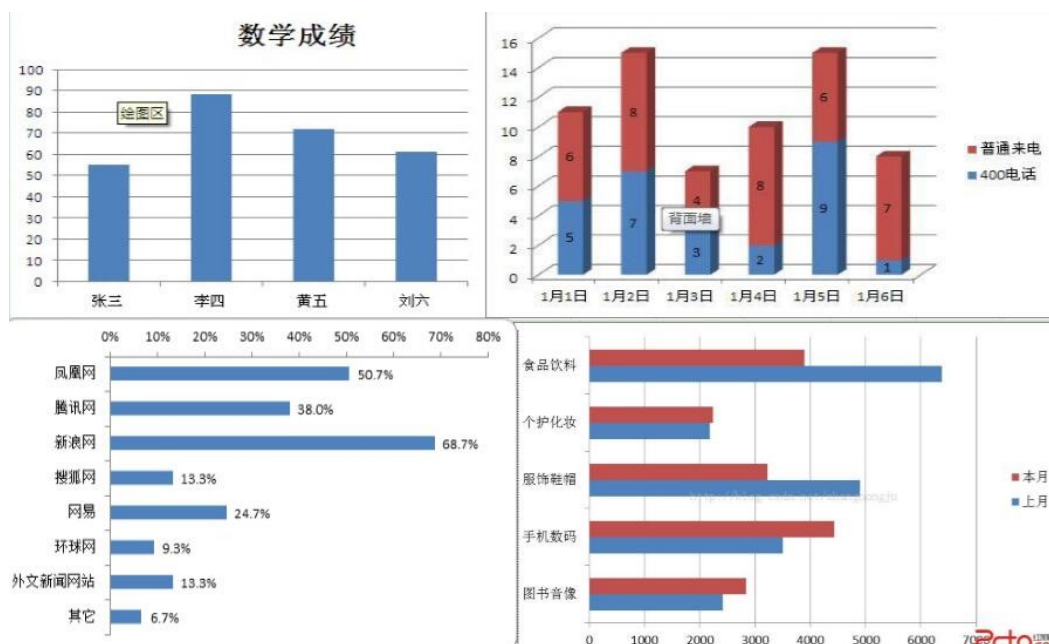


图 3-3

趋势比较常见的是时间趋势、剖面趋势，趋势要求自变量要有内在的连续型。趋势类图表的基本类型为折线图、散点图、条形图。如图 3-4 所示  $x$  轴为时间，自然具有连续型，图 3-5 是学历剖面的趋势，博士到初小也具有学历由高到低的递减，因此可以使用趋势类图表。



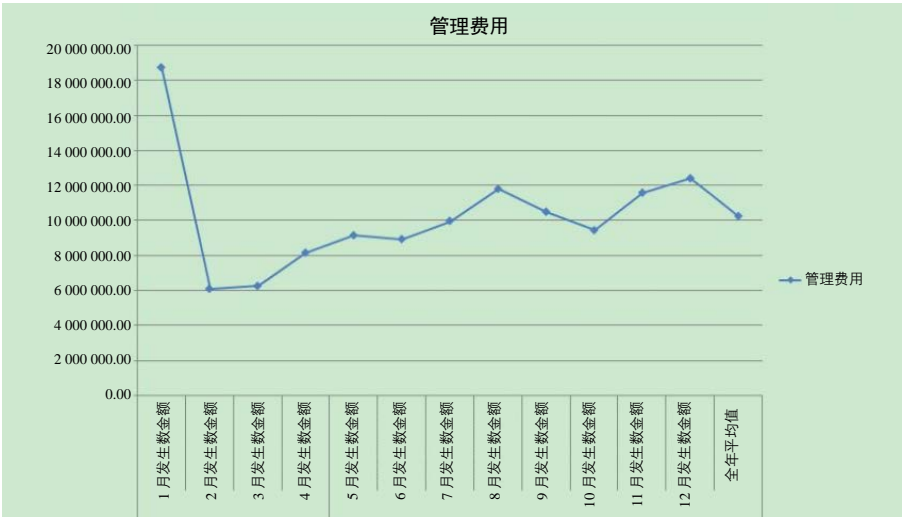


图 3-4

图 3-6 就犯了数据关系和基本图表类型不对应的错误，首先作者要表现的是一种趋势，虽然数据明显有下降的趋势，但是图的自变量是分类变量，从张三到胡适之不具有递增或递减的趋势，这幅图真正的关系是横向对比，应该采用表达对比关系的条形图、柱状图作为基础图形。

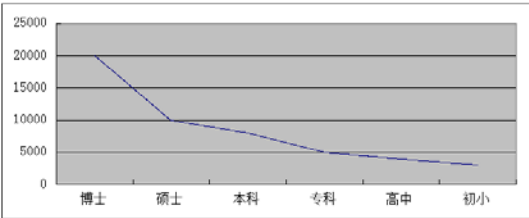


图 3-5

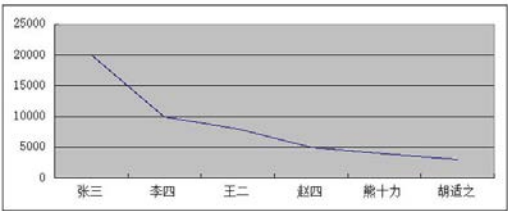


图 3-6

相关关系我们只讲单变量相关，如果是时间序列，则可能存在自相关关系，表示相关关系的基础图表为折线图和散点图。

分布图可以说上升到统计关系了，它反映的是数据点的分布特征，常见的有金字塔图（也可以作为对比关系的变形图）和箱线图，反映分布的统计特征值包括均值、众数、中位数等（见图 3-7）。

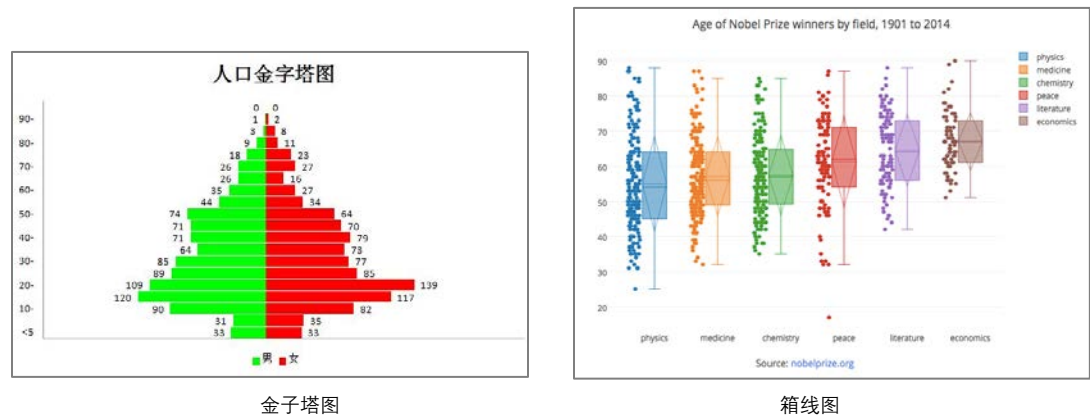


图 3-7

数据关系和基础图表基本包括上面讲的这些，上面的图表除了有时会错误选择基础图形以外，仍然显得不够商务，比较基础。

## 2. 配色

就数据分析工作而言，数据可视化可能是其最后阶段，而且也常常是工程师比较容易忽视的阶段，数据可视化牵涉甚广，而配色可能是大多数人认为的细枝末节。

大家习惯默认色，默认色就是偷懒，就是随便，随便就是不专业。另外有些时候我们意识到默认色实在太土，于是自己改了颜色，但是改的颜色不成体系，不是亮度不搭配就是色阶跨度太大等问题。

其实配色做得不好，一般人也不会明确地指出，但总会在内心里低看一眼。专业的配色总是能够在关键时刻拯救图表，让受众有一种愉悦的快感。从成千上万种颜色中选出合适的搭配颜色确实需要一定的功力，有时候颜色看起来能够愉悦眼睛，但是却让阅读更加困难。

细枝末节往往能体现一个人的专业水平和专业精神。

尽管颜色偏好这个话题主观色彩过于强烈，但我们还是可以从中找出专业的底线的。不可否认，有些颜色之间可以和谐相处，但有些颜色却相互排斥，因此随机选择颜色多少有点过于鲁莽。很多研究人员或艺术家的经验确实能够帮助我们识别一组和谐的颜色。其中比较常见的方法就是使用色环选取，人类早在 18 世纪就已经使用色环选取颜色。色环是光谱色从红到紫围成的一个环形结构，它应该是我们最常见的颜色选取方法之一。

模仿专业图表的配色是一种快速提高配色水平形成自己的配色体系的方法。配色体系：标志色、背景色、主图色系，标志色要醒目，比如《经济学人》的标志色为鲜艳的红色，《华尔街日报》则为纯黑色，背景色要浅，因为所有的主图色系都要在背景色上搭配，主图色系需要注意统一融合，同时控制颜色种类。我们这里推荐以《经济学人》《华尔街日报》等著名杂志或网站的图表作为配色模仿的标的。

如果要形成自己的风格，那选择颜色时也要有几点注意，首先注意颜色与语义的搭配，例如要用黄色表示香蕉的销量，用紫色表示葡萄的销量，用蓝色表示蓝莓的销量。《哈佛商业评论》研究发现语义与颜色结合紧密的图表更易读。其次要赋予颜色层次感，层次感当着眼于数据的重要性，需要强调的数据就需要给予更高的亮度和更鲜艳的颜色，重要性降低，同时亮度降低。

颜色有时候具有一定的心理暗示作用，绿色是一种冷静的颜色，给人以希望和稳定可靠的暗示，而红色往往能够传递激情和能量。PayPal 和 LinkedIn 使用蓝色暗示自己值得信赖，而 CNN 和红牛使用红色传递自己的能量。

最后，一旦选择一个色系，就应该固定下来不要轻易改变，稳定也应该是专业的题中应有之义。稳定要求在一段时间内保持自己的用色风格，让别人看到颜色就知道是你的作品，另外稳定还要求在一个项目内颜色的统一。

### 3. 注重细节

细节决定成败，一个人的用心程度往往反映在一些细节上，有时候技能专业与否倒是其次，关键是用心。专业图表的细节包括数学辅助和几何图形两类，数学辅助包括坐标轴、坐标刻度、坐标轴标签、网格、标签等；几何图形包括点、线、形状、阴影等。

我们看图 3-4 和图 3-8，图 3-4 处理坐标轴时犯了两个错误：Y 轴刻度标签需要更改单位（参看图 3-8《经济学人》的处理），否则看起来数字太长了；X 轴刻度标签同样处理不当，当标签太长时应该想办法缩小长度，而不是任其蔓延，另外由于标签处理不当，X 轴的刻度线也变成了丑陋的长线，《经济学人》的处理方法为在开头和中间的某个点显示完全标签，从而省略了内容，另外，《经济学人》还标注了图表的数据来源，这个细节足够证实你的商务素养。

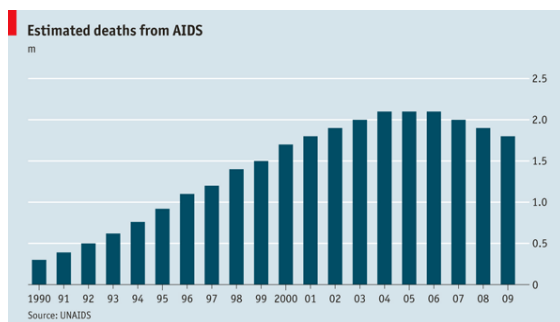


图 3-8

数据标签的作用是突出重点，当数据量较小或要突出某个数据点时可以使用数据标签，当数据量较大或表示趋势时使用坐标轴网格线标定数值。坐标轴网格线和标签同时滥用就不够专业，图 3-9 就犯了这种错误，图 3-10 是《经济学人》的图表，使用了坐标轴网格线就没有再使用标签。有人会问图 3-10 的右边不是数据标签吗？右边还真不是条形图的标签，那个标签是另外一组数据映射而成的。另外网格线要注意粗细、线型、颜色深浅，标签要注意形状、位置、颜色等。

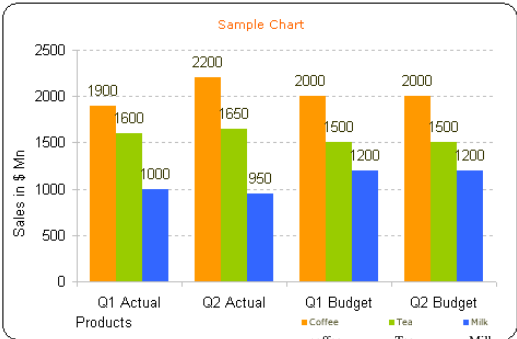


图 3-9

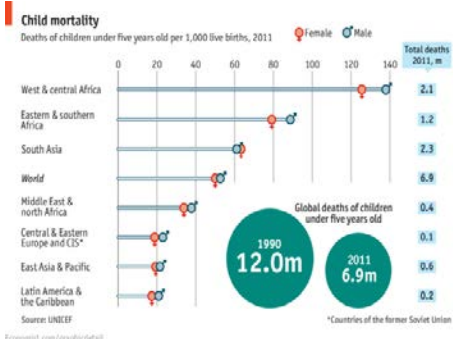


图 3-10

使用几何图形时要注意几何图形的颜色、种类、填充、边线，另外还要注意几何形状是否和其他标志作用重复等几个方面。图 3-11 除了线和形状的主次不突出的问题以外，还同时使用了方形、圆形、三角形、菱形等 5 种形状，作者使用这么多形状的目的可能是为了区分不同的成分，但是这一点已经可以通过线的颜色十分明白地区分，所以属于作用重复，同时增加了图形的阅读难度，正确的做法应该是统一使用一种形状，图 3-12 的做法就比较专业，主次分明没有功能重复的配件。形状的使用要注意控制种类数量及其他形状的搭配，折线要注意线型、平滑、粗细、颜色等。

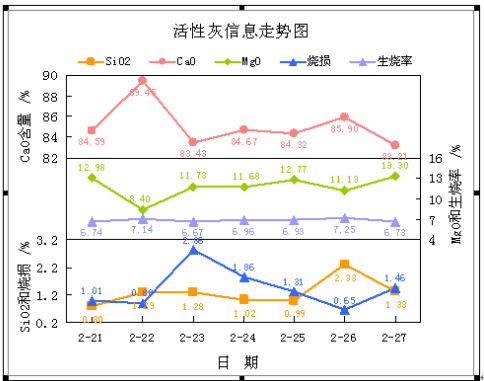


图 3-11

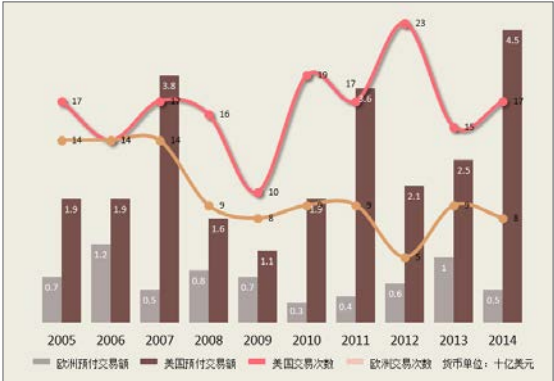


图 3-12

#### 4. 善于变形

图表能力提升的难点在于图的变形，有时候我们觉得饼图很难看，可是我们不知道怎么改变，其实只要记住饼图的核心就是对比同一体内的各成分的大小，关系不变其他都可以想象改变，我们就会发现很多种变形。万变不离其宗，从而将专业水平提升到一个新的高度。

另外，变形时我们还要考虑图表的结合，包括图与图的综合，图与表的综合，综合的成败是位置、形状、配色的搭配和统一。

5. 简洁是商务图表的底线

专业图表的另外一个特性就是简洁。做到简洁首先要学会组装，将多图组合变形为综合图，这样就丰富了图的信息，简洁包括主次分明和整洁，综合图表时容易失去主次分明的特性。如图 3-13 所示使用了颜色的深浅表示主次。

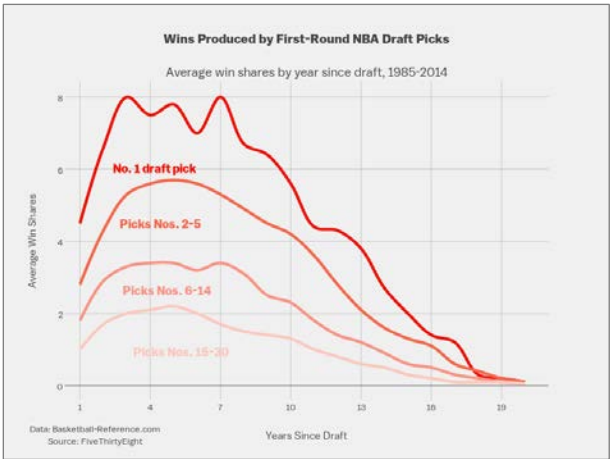


图 3-13

简洁的基础是整洁，下面是两个同样类型的图，但是为什么图 3-14 显得不专业，图 3-15 看起来更舒服（《经济学人》），除了配色导致二者水平的差异以外，前者的数据标签摆放得非常随意，而后者的数据标签非常整洁，用颜色做出了增减的区分，从而提供了更多的信息。

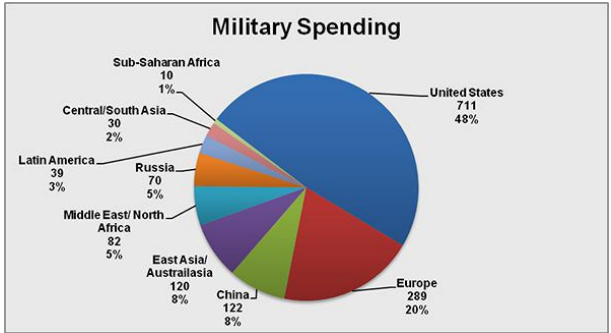


图 3-14

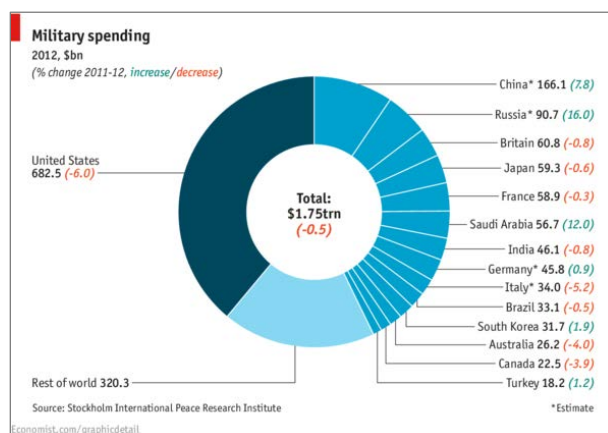


图 3-15

模仿使我们接近专业，专业主义的态度表明专业需要付出更多的精力和时间，而商务和职场就是一个专业化的场所，只有专业的人才能以令人尊敬的方式吃到肉，寻找自己心目中的完美，并把它一直重复，直到你的做法成为别人的标准，你就是所谓的专家。

### 3.1.3 那些你不知道的图表误导性伎俩

《定量信息可视化》这本书里曾经记述过这样的一段话，个人觉得对如今的可视化设计者们仍然有深远的借鉴意义。

设计本质上就是做选择，数据可视化的一些金科玉律能够帮助你选择较为合适的选项，但固执僵化的执行这些理论显然也没有必要，它们并不具有数学一样的严密逻辑，因此也非必由之路，正所谓过犹不及。因此要以怀疑的态度使用这些可视化原则。选自《定量信息可视化》

但有时人们对有些图表已经形成特定的解读方式，对这类图表做出改变时就要小心谨慎了。

#### 1. 基线要从零（最小值）开始

柱形图、折线图用高度塑造数据，比如柱形图的短柱代表较低的数据值，长柱代表较高的数据值，因此柱形的高度比较代表着数值之间的比较，如果你的柱形图不是从零或者对比基准点开始，会发生什么？

从图 3-16 中可以看到，如果柱形图的基线从零刻度开始，一切正常，100 是 50 的 2 倍，但随着基线的提高，错觉产生了，越来越强烈，导致受众以为第二个柱子是第一个柱子的  $n$  倍，这就为受众造成了阅读障碍，同时也暗示了错误的信息。因为可视化的信息最是从视觉认知开始的，只有在必要时才会查看标签和数据轴。

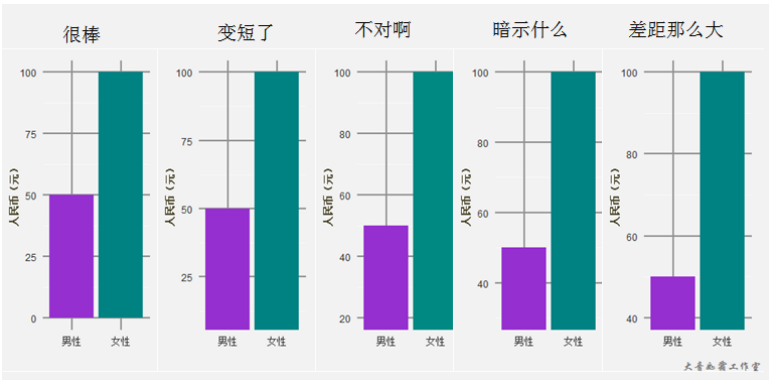


图 3-16

所以柱形图（条形图）的基线必须从零开始。

犯这种错误的图表俯拾皆是，有些图表其他方面做得非常专业，但是失去了可视化准确的灵魂也只能是华而不实。图 3-17 所示是个错误的例子。

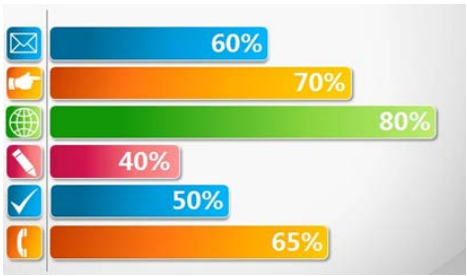


图 3-17

从图 3-17 中可以看到，80%的柱子长度是 40%柱子长度的 3 倍多，这显然是不符合逻辑的，就是故意夸大差距的显著程度，注意不要把单次差距和差异显著不显著混为一个问题，请大家一定要坚守自己的底线。

除了柱形图以外，还有后面要讨论的双轴图，也是容易恶意夸大或者误导受众的图表，而且双轴图的粉丝很多。

## 2. 故意调节数轴的最大值

上面说的是改变数轴的最小值的例子，还有一种方法是通过改变数轴的最大值来操控数据的趋势，你可能已经想到了，如图 3-18 所示。



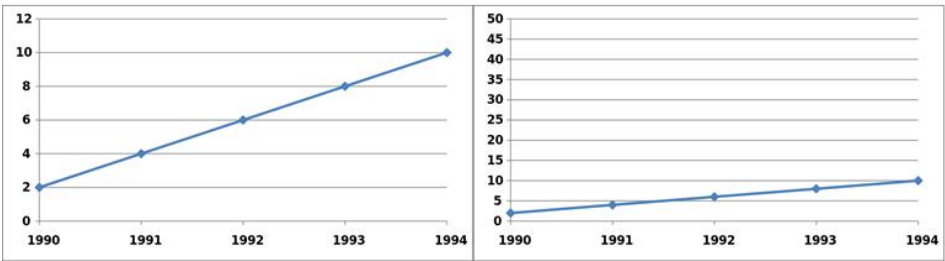


图 3-18

这种方式除了通过改变数轴的最大值以外，还可以通过调节图的宽度来达到目的，不信你可以将一个折线图保持高度缩小宽度以后看看斜率的变化。

坐标轴真的是大有学问可作，你还可以改变坐标轴的刻度以减小或夸大趋势的波动，只是大家使用时一定要小心。

### 3. 苹果公司使用饼图的秘密

有人说要尽量避免使用饼图，可能有一定的道理，但是个人认为饼图是一种妇孺皆知阅读毫无障碍的基础图表，巧妙使用也许能达到意想不到的效果，但需要注意的是不要过度切分饼图（见图 3-19）。

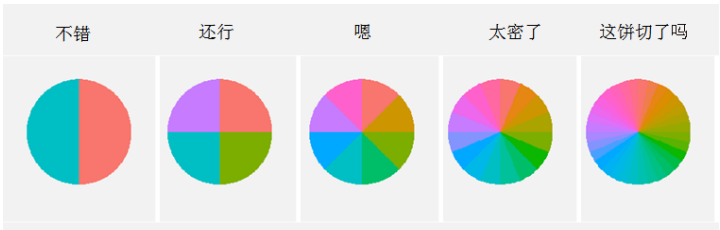


图 3-19

怎么把握度呢？当一些小类之间的大小肉眼已经分不出比较特性时或无法分辨所代表的类别时，就要考虑是不是过度分类了。如图 3-20 所示就是一个过度的例子，它来自维基百科。

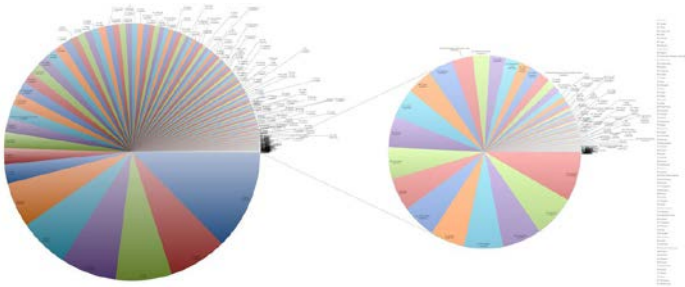


图 3-20



为了展示所有国家的比例，错误地选择了饼图，分类太细。解救办法有两种：将一些小类统一划分为一个大类；使用其他替换图形，饼图的替换图形有很多种，例如环图、条形堆积图、面积图、马赛克图等，如图 3-21 所示。

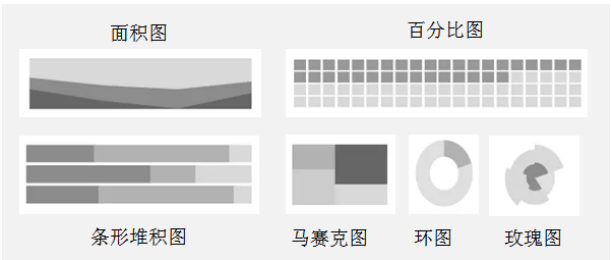


图 3-21

还有一个错误比较低级，即不把饼图当成整体来切分，如图 3-22 所示。做点图时也会碰到很多点聚集在一起的情况，点重合在一起无法分辨，如图 3-23 所示。

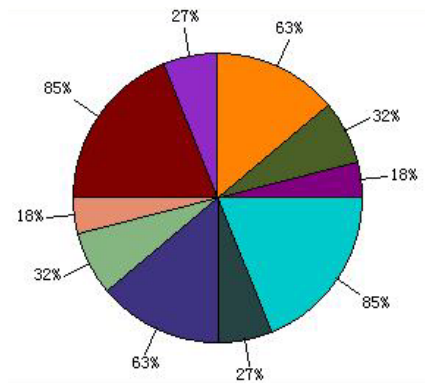


图 3-22

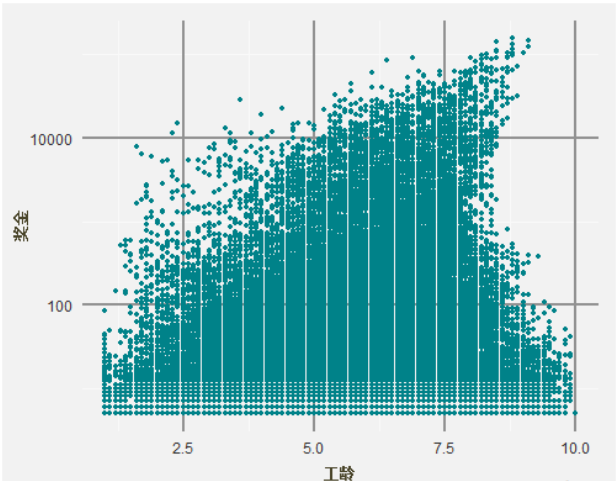


图 3-23

处理点重合的方法比较多，例如让点随机摇摆一下、缩小点的大小、改变透明度、扩大图的大小等。

4. 注释不清晰

数据可视化可以看作是对数据的再编码，你可能通过形状、颜色、大小等对数据塑形，因此设计者有责任帮助受众顺利解码，必然需要对自己的代码注释清晰，比如坐标轴所代表什么、数据单位等，很多人会使用这个伎俩来改变你阅读图表的感受，如图 3-24 所示。

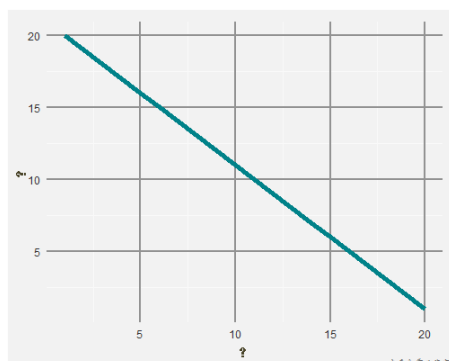


图 3-24

在设计时要注意，如果规则是有关“解码”的，最好不要打破，因为这将影响受众的阅读质量，甚至误导受众，至于其他类型的经验，倒无关紧要。

人类对数字的敏感导致造假者有机可乘，比如某化妆品广告说保湿提高 70%，保湿是怎么测量的？它是和谁对比的？基于人类对数据的敏感，于是有了各种诱导的手段，图表如是，营销也如是。

### 3.1.4 如何快速解构著名杂志的图表

迅速提高自己的水平就要“站在巨人的肩膀上”，数据可视化领域同样存在一些巨人等待我们去模仿和超越，不妨从著名商业杂志《经济学人》的 *Graphic Details* 开始学起，这个栏目带我走进了专业、商务的可视化领域。

仅仅实现某种图表和掌握某项技术还不能让你变得足够优秀，这类教程多如牛毛，往往可以快速掌握。掌握了一个领域的技能只能说明拥有了跨入门槛的能力，利用时间和持续的精力为自己划定专业的领地范围，这才是一个人的技术成长之道。

专业往往和个人风格紧密联系在一起，当然这种个人风格不是大众具有的错误，而是在一个领域内只有极少数优秀的人才具有的特质，这种风格往往是划分专业和大众的标准，除了技能以外，在自己身上集中越多的个人风格，你就越专业，但请不要忘记这些风格是以巨人为楷模的，而不是自身心血来潮的标新立异。所以在这个章节你阅读到的不仅是实现某种图表的技能，更重要的是你将看到很多商务的、专业的、与众不同的东西，技能忘记了可以临时学习，可以抄代码，风格失去了或者不能形成专业的思维，就永远只能做些稀松平常的事情。

《经济学人》敢为天下先，对世界各国的创新能力进行了一次综合排名，声称康奈尔大学动用了 79 个指标对全球 140 个国家或经济体进行了创新能力排名，这些名头只是为了让你相信这次排名不是数字骗局，而是一次专业的量化行动。虽然我极力反对排名，因为排名本质上是一个推荐系统，具有很大的主观因素。但是仍然不妨碍我们从这篇文章中的图表上学习一些专业的知识，另外不无遗憾地告诉大家排名前五的是瑞士、英国、瑞典、荷兰和美国，文章竟然指责我国虽然拥有大量的专利，但就创新质量而言仍然远远低于美国等发达国家。

图 3-25 是《经济学人》量化排名的气泡图，毫无疑问，这个老牌商务杂志又一次展现了其专业的风范。其中不乏可圈可点之处，真是入门学习的样板教材。模仿一幅图首先要有能力快速解构图形，然后重新组装整合。有时候我们经常碰到别人指责自己的作品太低级，但对方又说不出所以然，这就是缺乏专业素质的表现。顺利完成图表的解构之后，才能从专业视角对别人的作品内容分门别类地品评。这里使用世界卫生组织的数据模仿了一幅《经济学人》的创新能力气泡图，如图 3-26 所示。

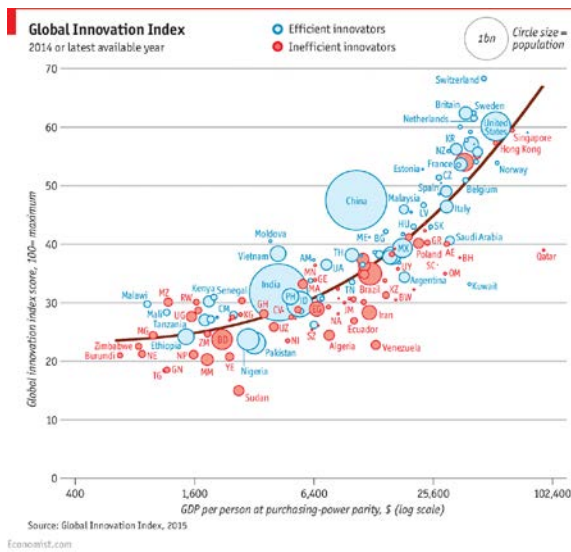


图 3-25

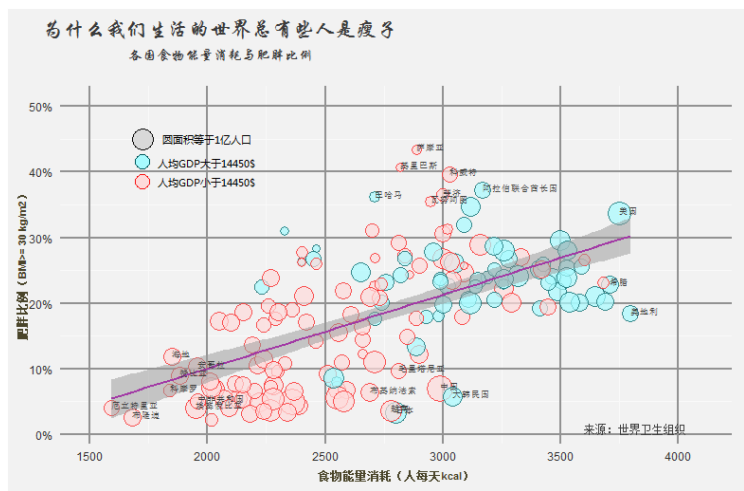


图 3-26

常见的绘图设备或软件都会将一张图表分为两个区域：图表区和绘图区。后者在前者的区域内划分出了独立的范围（请记住独立范围的含义），另外绘图区受坐标系限制和定位，图表区不受坐标系的管制。

（1）图表区：标题、附带说明、标签、几何形状等。

（2）绘图区：坐标体系、几何形状、统计变换、标签、附带说明等。

先说看起来比较简单的图表区，图表区就是整个图表除去绘图区占用的部分，即图表边界，图表边界里的元器件大多和文字相关，包括主副标题、坐标轴名称、作者标志和数据源标注。严格来讲只有坐标轴名称必须位于这个位置。主副标题、作者标志、数据源标注可以独立在图表之外，也可以放在绘图区，甚至有些要求简洁而不那么严谨的作者直接忽略这三项，但是任何一个有专业素养的人都不会忽略坐标轴名称，另外坐标轴刻度单位也会跟在坐标轴名称后面。

### 1. 图表区

（1）主副标题

（2）坐标轴名称

（3）作者标志

（4）数据源标注

（5）图例

（6）背景

除图表区边线以外（很多情况下会省略边线）的都是文字。与文字相关，可以形成风格的地方主要包括字体、大小、颜色、位置这几项，这些与图表区的搭配可以显示一个人的功底水平，大家要注意中文应该用什么样的字体，英文和数字又应该使用什么样的字体等细节，以形成自己的风格，当然这个区域评价图表也应该从这些方面着手，另外一个更重要的点就是涉及文字的内容一定要做到言简意赅，而不是率意而行。在图表的地界上，虽然文字成为配角，但这里的片言只语更应该做到精炼而达到言尽其意、画龙点睛的效果。

附带说明一般包括作者标志和数据源标注，当然作者标志之类的细节除了使用文字以外，也可以使用特殊的色块或者 LOGO 为自己的图表打上烙印，这是可以体现浓厚的个人和公司色彩的地方，《经济学人》采用了两种方式标记自己的图表，首先每一张的左上角有公司的标志色块：红色色块（几何形状），其次在图的底部有公司的网址，其精细入微不禁令人叹服。

数据源标注体现的不仅是个人风格，也是作者专业素养的综合体现，合理地使用数据源标注体现了数据的权威性，不容置疑。退一万步讲，即使有问题也可以追本溯源，同时也给其他人再造作品的机会，什么是专业，这就是专业！当然数据源标注不一定在图表区标注，也可以在绘图区标注，如图 3-36 所示，但最好的处理方式是放在图表区，除非图表区已经安排了其他内容。

图表区还有一个分歧很大的元件：图例。一般有四种方式处理图例，其一，如果图表本来就很简单则可不要图例；其二，将图例放在图表边界内，这是很多软件常见的方式，Excel 和 ggplot2 都是默认放在右边界的；其三，将图例放在绘图区的空白处，但这在 R 里很难做到，我们的方法是

原有的默认将图例去掉，而在绘图区的左上边界添加几何形状仿制一个图例；其四，直接在图中的几何图形上标注文字，这也是我们公众号常见的处理方式，后两种处理方式最节省空间。图例一般是众多绘图软件忽略的元件，做得不够精致，缺少很多自由的可发挥空间，如果有时间，建议去掉默认的图例，自己用几何形状仿制一个图例。

图表区最后要说一下背景，背景的填充有三种：其一，透明背景，不做任何填充，这方便将图添加到其他图上，避免了不必要的遮挡；其二，就是最常见的纯色填充，这个背景色是风格的标志之一，这里所说的风格指的是别人一看就知道是你和你的小伙伴绘制的数据图表，所以要慎重，《经济学人》常用的填充绘图区背景色有白色、灰色和天蓝色，如图 3-37 所示；其三，图片和纹路填充，有时候合适的纹路填充可以给人一种图表跃然纸上的感觉，而图片填充多是为了绘制图上图。

需要提醒大家的是，不要以为填充了图表区的背景整个图片的背景就都被填充了，绘图区是图表的独立小区域，在很多软件中它的背景需要另作一次填充。绘图区讲究的不是个人风格，这里要求科学准确地表达数据所反映的规律，这才是核心。绘图区一般包括坐标系、几何形状、标签、网格体系和背景等。

## 2. 绘图区

- (1) 坐标系
- (2) 几何形状
- (3) 统计变换
- (4) 标签
- (5) 网格体系
- (6) 背景

坐标体系常见的有三种：常见的有二维或三维坐标、极坐标和叠加坐标。二维和三维坐标最常见，经纬度坐标可以算作二维坐标的一种；极坐标的代表作就是“提灯女神”的玫瑰图以及戴布拉图（又称雷达图），还有屡次遭人诟病的饼图；第三种叠加坐标的例图是数据流图，也称 Sankey 图，它要求每一个柱子位置是一个坐标体系，而柱子的高矮又是一个坐标体系，这样解释大家可能无法想象，举个例子，ggplot2 就是典型的单坐标体系，如果要在 ggplot2 绘制的地图里的某个省上添加一个小柱形图或饼图那是很难办到的，因为你的柱形图是另外一套坐标系，不遵循地图的坐标体系，否则会很丑陋，ggplot2 要求所有的图层都遵循一个坐标体系，这项任务需要借助其他包完成。

几何形状均是数据的映射，点、线、面从形状、大小、高矮、填充色、透明度、边线、粗细等都可以映射到数据，只不过这些属性有些宜于反映连续变量而有些则应该用于分类，但是切忌功能重复，所谓功能重复就是一个数据维度用两种或两种以上的几何形状反映，这样就增加了读图的难度。我们为什么作图，就是宜于受众接受数据所反映的规律，所以任何有碍于受众理解的东西都应该避免。几何形状也会出现重叠的现象，处理方式有两种：改变几何形状透明度，如本例；让几何形状随机移动一点点位置，但效果不佳，仅仅适宜于轻度重叠。

统计变换是帮助受众理清规律的一种辅助方式，当然也常常沦为诱导受众做出错误决策的方式。

统计变化常见的有趋势线（拟合线）、误差线等方式，但使用时一定要确定趋势是否存在，进行相关的统计检验，否则就是弄巧成拙。另外，统计变换不能为了变换而变换，要遵循实际，有业务支撑，如不能使拟合的温度相关曲线延伸下去达到绝对零度。

绘图区的标签主要分为两种：坐标轴刻度标签和几何形状标签（哦，偶尔还会有图例标签）。坐标轴刻度标签极其重要，没了它就成了瞎子摸象，而几何形状标签就比较随意，没有必要将所有气泡代表的国家名称都标出来，给予某些点标签是因为它们反映了一种现象或规律。这就是几何形状标签重叠的一种处理方式，另外可以通过随机移动进行处理（jitter）重叠。坐标轴刻度标签重叠可以通过三种方式缓解：① 倾斜角度；② 上下位置错开；③ 放大刻度等同于减少标签或缩短标签，比如如果是年月日，只在年月发生变换的地方显示完整标签，其他仅显示到日，《经济学人》比较喜欢使用这种处理方式，如图 3-27 所示。另外坐标轴标签是跟着坐标轴刻度线（ticks）的，坐标轴刻度线除了拥有线的一般属性以外，还有两种标注方式：向内和向外，建议如果是连续型数据坐标轴刻度就向内，分类型数据就向外。

网格体系比较简单，仅仅包括主网格线和次网格线，不要过度使用网格线，最好有主次之分，网格线的作用是帮助定位刻度，多了反而增加阅读难度，所以例图的主次之分很明显（灰白两色），既不会增加难度也可以迅速定位。网格是由几何形状线组成的，它可以用线型、粗细、颜色形成个人风格，个人觉得这种辅助阅读的工具不必搞得过于吸引眼球，否则就适得其反了。



图 3-27

绘图区背景和图表区的内容一样，虽然我崇拜的对象《经济学人》经常将图表区背景和绘图区背景区分开来，但个人倾向于和图表区保持一致，避免给读者造成太多的疆界隔离。

到这里，我们就基本学会了应该从哪些方面解构一张图表，然后品评它、模仿它，最后超越它。另外，知道了这些元器件，可以帮助你建立起品评体系，而不是简单的丑和美之类的于事无补的评价。

## 3.2 ggplot2 包：一个价值 8 万美元的态度

ggplot2 是 Hadley Wickham 的心血之作，如果你已经深入地学习了 R，那一定使用过三大神器：dplyr、reshape2、ggplot2。它们的作者就是这个号称“革新了 R 的人”，现在是 Rstudio 的首席科学家。Hadley 出生于统计学世家，爸爸、姐姐都是学习统计出身。

ggplot2 的绘图原理来源于 Leland Wilkinson 的名著 *Grammar of Graphics*，主要认为统计图形就是数据到点、线、面（方块）等几何图形及其属性（颜色、大小、形状）的映射，将映射和一些统计变换绘制到某种坐标系中就是统计图形，Hadley 读完后高山仰止，直接取了书名的缩写作为开发包的名字 ggplot，Hadley 一般开发一个升级包后，就会在原来的名字后加“2”，这就是 ggplot2 鼎鼎大名的来源。

ggplot2 旨在吸取 R 另外两个绘图体系 base 和 lattice 的优点，去伪存精，成为更加易用、专业、艺术性的绘图系统，现在已经成为 R 三大绘图体系中最出名的一个。但是这些都不能说明 ggplot2 是一个有态度的包，所谓态度就是对某一事物深深的执念。ggplot2 摆出了令无数用户扼腕叹息的态度，它做了至少两件如今还为人诟病的事情。

（1）不支持三维图形

（2）不支持双 Y 轴坐标（不支持双轴图）

没错，ggplot2 里不支持三维图形，甚至图形的立体效果都没有，原因很简单，如果支持三维，就违背了它的绘图哲学，它认为对象（几何形状）可以按照图层一层一层叠加到坐标轴固定的画布上，这和 Photoshop 比较相似，因此，ggplot2 绘制三维图形只能用等高线、渐变热点绘制，Hadley 认为数据可视化展示的是规律而不是数据本身，绘制山头的高度变化时，没有必要绘制出整座山头，否则会影响受众的注意力。话虽如此，但当使用 ggplot2 绘制数学函数或物理场时就蔫了，不过，对于一般的市场研究人员又有多少次会遇到这些情况？想想也就无须求全责备了，R 从来不是一个包就一统天下的语言，R 语言的处理方式就是不同的问题就找相应的包，包的功能和问题对应。

不支持双 Y 轴坐标争议更大，我们还是请 Hadley 自己给大家解释一下：

（1）如果使用双轴，那么整个映射过程就不可逆了，也就是说，图中的点很难对应到数据空间（作者注：原来一个坐标轴，图上的点很容易定位到数据集，如果使用两个轴，因为图上的点可能来自不同的数据集或同一个数据集不同标度的转换，这就是多对多的关系，无法逆转，只能通过其他维度加以区分，比如颜色）。

（2）与其他变通方式相比，双轴增加了阅读难度，参见论文 *A Study on Dual-Scale Data Charts*。

（3）对于坐标轴的标度很难有标准，这样就容易引起误解，比如相同的温度，使用摄氏和华氏两种单位的坐标轴就会显示出不同的意义，其实它们表示的温度是相同的，诸如此类的还有汇率等。

（4）双轴的观点比较主观，有了两个轴为什么不能来三个、四个。

最后，终于有一个网友把 Hadley 惹毛了，他就愤怒地在 StackOvenflow 上评论说，如果想要我在 ggplot2 里添加双轴这个功能，不是不可以，需要支付 8 万美元的报酬。我还能说些什么呢，真是一个执拗的数据科学家，但是执拗的人坚持的不一定不是真理，只是不愿意迎合大众的口味。

看了 Hadley 的评论我这里还真要为他打抱不平，其实一个人坚持自己的态度和原则已经很难，何况是面对大多数人的攻击时仍然坚持非常正确的理念呢。下面我们就从几个方面说说双轴图的原罪，双轴图的意义无外乎三点：

- (1) 比较差值（高度差，面积差）
- (2) 比较趋势
- (3) 节省空间（哦，这个不能算吧）

首先，比方说我们将两列数据绘制在单轴上，发现它们接近平行线，这应该是它们本该具有的状态，但是如果两条线的单位不同，你觉得它们之间的差值还有比较的意义吗，比如日元和美元？另外，如果我们在图上添加一个轴会发生什么变化呢？如图 3-28 所示，右上图是面积图，你很容易发现它们之间的差值缩小了，所以如果是双轴不同标度的折线比较差异是不靠谱的（很多国外记者使用这样的招数）。那么既然差值不靠谱，趋势比较总可以吧？

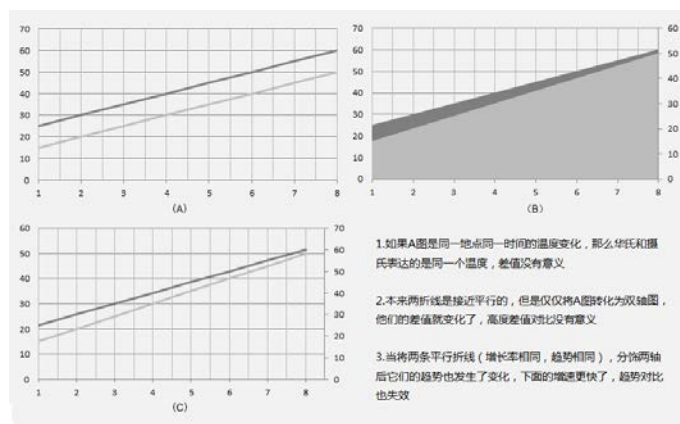


图 3-28

答案仍然让人很失望，趋势比较同样会误导受众得出错误的结论，同样是图 3-28 的数据，左上图原数据在单轴同标度下是两条接近平行的折线，它们的增长率是一样的，趋势是相同的，不难看出，如果绘制成不同标度的双轴图，那么两条线就不是平行线了，斜率变了，趋势变了，一切都是人为操纵的结果了。综上所述，将两个标度不同的图合为一张图根本完不成任何对比效果，反而会误导受众得出错误的结论，这些恐怕就是出乎领导们意料之外的吧？绘制双轴图唯一的好处就是节省了空间，但这种以误导受众为代价的节省空间，哪一个有态度有原则的数据分析师会接受呢？

下面就是《纽约时报》利用双轴图误导读者的一个经典例子：An Old Formula That Points to New Worry，文中给了一幅双轴图，如图 3-29 所示，用来说明一个时间拟合模型和纳斯达克 500 股指的



拟合趋势，我们放下这个模型的统计学漏洞暂且不提（很多），我们评价拟合时首先评价的是对趋势拟合得好不好，其实可以按照上述方式，调节其中一个轴来控制时间模型的拟合结果，想让它看起来很棒，就调一下其中一个轴，改变趋势斜率，想让它看起来很糟糕，也可以使用同样的方法。

文章还不无廉耻地说随着时间的推进模型拟合的结果和股指之间的差值更大了（即中间空白），是的，看过上面的分析你就知道，它们之间的空白根本没有意义，想留更多的空白，调节一下轴标度就可以了，真是非常“奇葩”的思维。

不可否认，作为一个数据可视化设计的工作者，你可能会被要求将两个不同标度的图放在一起，方法是这样，你可以给他讲双轴图的缺点，如果他不听，就让他使用双轴图去做决策吧。

只要有人的地方就有争执，因为人总是沉迷于理性的选择。其实争论毫无意义，就 R 的绘图包而言，有些人就会陷入选择困难症，通常的问题是应该使用 R 基础绘图包、lattice 还是 ggplot2 等其他绘图包？其实真正在工作中，每个人总有各自喜欢的方法，但是当你的任务必须由某个方法完成时，你又不得不放弃自己的真爱而选择临时应变的策略。我通常情况下使用 ggplot2，但是为了点对点地快速讲解某个规律，可能临时使用基础包里的绘图函数。没错，现实就是随机应变，没有太多假设，争论过程中有所推进才是最有意义的理性活动。

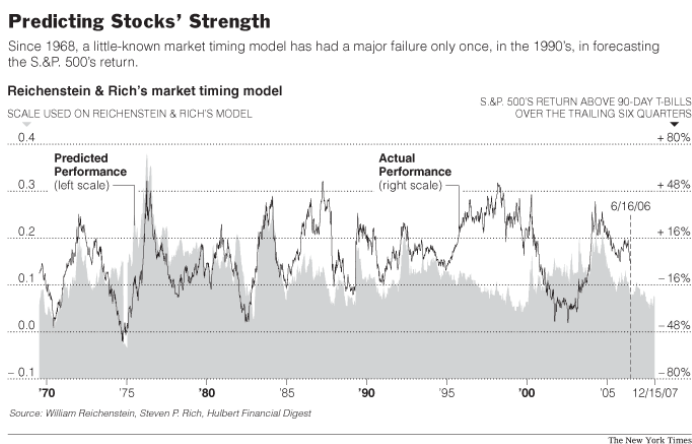


图 3-29

R 的基础包在一般情况下用于快速生成探索报告，而不必纠结于一些不必要的东西；ggplot2 通常用于印刷级的数据可视化，比如本书中很多图表都是使用 ggplot2 精心绘制的，也算是一种选择方式吧。

### 3.2.1 一张图学会 ggplot2 包的绘图原理

ggplot2 的绘图原理和 Photoshop 比较像，将所有对象看成图层，可以使用相关的语句一层一层地添加。另外，它将图层和数据分离，也就是说，你可以通过一个数据添加所有图层，当然也可以

通过多个数据集分批添加图层，很多人都将迅速绘出比较美观的图表视为 ggplot2 优于其他基础包的地方，但我不这么认为，因为 ggplot2 的默认主题并不能保证图表表达达到印刷级的审美，如果审美要求较高，ggplot2 也同样需要烦琐的修改，有趣的是，它能够把很多相似的主题打包在 theme 函数中，可以重复使用，这种打包方式使其看起来要比其他包简洁得多。ggplot2 真正的优势是它融入了一些常见的统计变换，这一点比基础包方便得多，因为在基础包中，你需要将一些统计变换预先计算并存放在数据集中，比如添加置信区间等。

上面的章节我们已经学习了解构图表的方法，这样我们学习任何绘图的方法都将简单很多。任何图表的元素基本包括几何对象、图形属性和统计变换，ggplot2 也不例外。在 ggplot2 中，这三大类元素通常由三类函数掌管。

(1) 几何形状：一般由“geom\_”系函数掌管，将数据映射为一定的几何形状，比如点、线、面等。

(2) 图形属性：一般由 aes 函数控制，包括坐标系、几何形状的图形属性（大小、颜色、填充）等。

(3) 统计变换：一般由“stat\_”系函数控制，主要用于展示变量的统计特征，比如拟合曲线、置信区间等。

上面最需要解释的就是“aes”，它是英文单词 aesthetic（美学）的缩写，这里应该将其翻译为数学上的映射，Hadley Wickham 自己解释为“a mapping of variables to various parts of the plot”，直译为“变量到图形部件的映射集合”，我的理解是 aes 用于设置控制图形属性的变量集合，虚化一点说，描述或指定图形和数据之间关系的设置都需要在 aes 里完成。

不能光说不练，想太多没用，我们不妨用上一章节的气泡图将这些要点一一验证一下，加深理解。《信号与噪声》作者纳特·席尔瓦（Nate Silver）是我最为敬仰的数据工作者之一，他也是 FiveThirtyEight 数据新闻网的创始人。在《信号与噪声》中有一幅有关健康的数据分析图，作者用以说明全球各国食物热量消耗（卡路里消耗量）与肥胖比例未表现出相关的统计特征，我们就用这份数据学习一下 ggplot2，重新研究一下食物热量消耗与肥胖率的关系，冒险犯一次翻案主义错误，同时使用 ggplot2 包绘制一幅印刷级别的图表，向读者传递我们的发现。

首先，读入数据，对数据简单探索一下，了解数据结构，如下所示。

- 数据探索

```
1 Obesity <- read.csv(file = "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数据可视化说起//data/Obesity.csv", header = TRUE, sep = ",",  
stringsAsFactors = FALSE)  
2 summary(Obesity)
```

数据共包括 7 列 4 个连续变量（人口数量、每人每天热量消耗、肥胖比例、GDP）和 3 个字符变量（国家汉语简称、国家 ISO 码、GDP 分类），而我们研究的是变量 Obesity 与 Kcal 之间的关系，不妨先制作一个散点图观察一下，代码如下所示。

- 散点图

```
1 if (!suppressWarnings(require("ggplot2"))){
```

```

2 install.packages("ggplot2")
3 require("ggplot2")
4 }
###开启画板
5 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, colour = GDPmean))
6 p
###添加散点层
7 p <- p + geom_point()
8 p
###添加折线层
9 p <- p + geom_line()
10 p

```

载入 ggplot2 包后, ggplot 函数除了开启一个画板外, 第 1 个参数指定数据, aes 用于指定控制图形属性的变量(映射关系), 比如纵横坐标、形状大小、颜色等, 我们指定了纵(Kcal)横(Obesity)坐标, 几何形状(点)的颜色属性用变量 GDPmean 指定, 这里的 aes 函数指定的变量是全局变量, 也就是说控制所有几何对象相应的图形属性; aes 函数在后续的句子中也可以设置, 但这里的 aes 在级别和权威上属于最高级。

效果如图 3-30 所示。

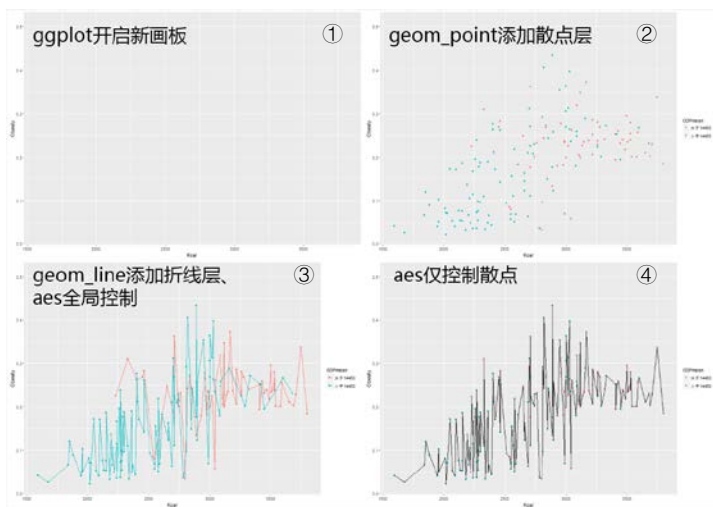


图 3-30

图 3-30 中第一幅图仅仅画了一个空画板, 也就是最底层的图, 所有后续图层都是叠加在这一图层之上的。使用 geom\_point 函数给原来的图层 p 加上散点(几何形状)图层, 然后不妨使用 geom\_line 函数再添加折线。默认设置下, ggplot2 绘制的散点图就完成了(恩, 还添加了折线), 下面先讲一下 aes 的级别和权威, 再进行美化。

- aes 的级别和权威

```
####在全局函数 ggplot 中设置 aes
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, colour = GDPmean))
2 p <- p + geom_point() + geom_line()
3 p
####在其他函数中设置 aes
4 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity))
5 p <- p + geom_point(aes(colour = GDPmean)) + geom_line()
6 p
```

在全局函数 ggplot 中的 aes 设置图形的 colour 属性后,我们发现 ggplot2 自动把数据按照 GDPmean 分为两组,两组的散点和折线使用了不同的颜色,说明这里的 aes 管着所有几何形状(图中散点和线)的属性,如果后续添加其他几何形状,则仍然按照 GDPmean 分组显示颜色,而且后续无法改变,这里属于全局控制。

如果去掉 ggplot 语句中 aes 设置的 colour 变量,迁移到 geom\_point 函数里,就变成了仅控制散点的颜色,折线的颜色并不受其控制,属于局部控制,这样我们基本上就理解了 ggplot2 全局控制和局部控制的异同了。

以上我们了解了 ggplot2 的图层概念(geom)、控制图形属性的映射(aes)及 aes 的全局和局部控制。有读者可能要问怎么绘制条形图、柱形图、折线图、雷达图、饼图等,它们其实都是通过 geom\_ 的后缀设置或通过坐标体系设置实现的,暂且按下不表。

如果一个人不能以庖丁解牛的方式分析一个问题,要么他是具有大局观的“领导”,要么他还是个门外汉。

下面就从图表常见配件上改造上面的散点图,一方面使其更加准确地反映数据集 Obesity 中的规律,一方面提高它的颜值,代码如下所示。

- 将散点图改为气泡图

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
  geom_point()
2 p
####scale(标尺)的概念
####设定圆大小标尺
3 p <- p + scale_size_continuous(range = c(1, 13))
4 p
####设定 x 轴的标尺 0~4100
5 p <- p + scale_x_continuous(limits = c(0, 4100))
6 p
####x 轴刻度线刻度为 300
7 p <- p + scale_x_continuous(breaks = seq(from = 1500, to = 4100, by = 300))
8 p
```

效果如图 3-31 所示。

在进行绘制之前我们对数据稍微改造了一下，由于 Population 变量里有极端值存在（中国和印度），我们对 Population 变量进行了对数变化，缩小了最小值和最大值之间的差异，如果不做对数变换（去掉 log），你会发现图上只有两个圆比较大，其他圆之间没有区分度。在统计中我们经常对数据做一些统计变换，包括整除、整乘、对数、指数、均一化或标准化等，以使其适应某种统计方法和应用情形。

所谓气泡图，就是将数值型变量映射为散点的大小，这里通过 size 将 Population 变量映射为散点的大小，散点图就变成气泡图了。通过 scale 函数设置圆的大小变化区间，scale（标尺）又是一个需要花费时间理解的函数，一般的映射都有一个设定标尺的问题，也就是说 aes 里设置的图形属性都需要通过 scale 设置单位，例如坐标轴有坐标轴刻度，颜色有深浅的单位，刚刚设置的圆大小也要设置标尺单位，比如这里我们指定了参量 range，即设定了圆的变化范围，将最小的 Population 映射成的圆的大小设置为 1，最大的 Population 映射成的圆的大小设置为 13，其他数据点根据比例关系进行调整就可以了。

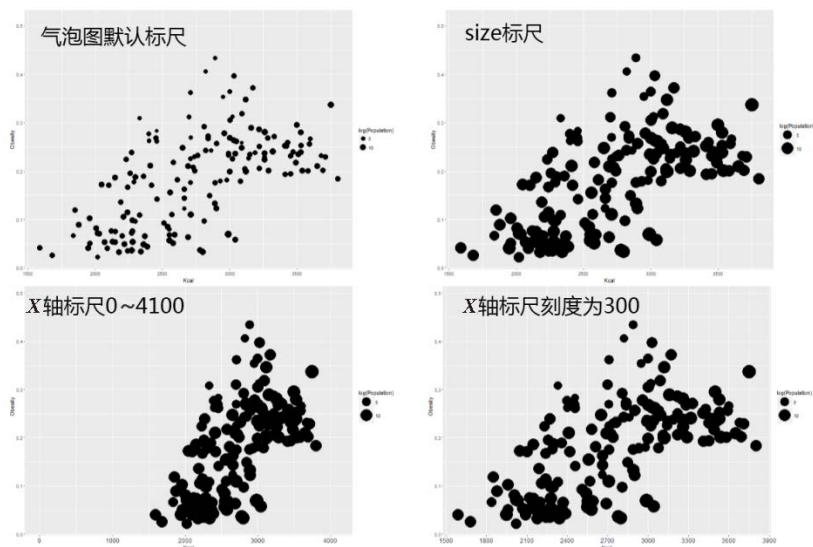


图 3-31

比如图 3-31，ggplot2 默认情况下 X 轴是从 1500 开始的，可以通过 scale\_x\_continuous 函数里的 limits 参数将其设置为从 0 到 4100；当然仅仅设置最大值和最小值，并不能满足我们的需求，有时还要设置到具体的刻度（或者更改刻度标签），那就需要设置 breaks 参数，在设置之前首先要生成一个数列，告诉 ggplot2 每隔多少显示一个刻度和刻度标签。

所有衡量差异的东西都要有一个尺度，那么这个尺度就是标尺（scale），标尺设置的方式有很多，只是表达方式不同，最终规定的都是从一个级别到另一个级别要发生多大的变化。

## 探寻数据背后的逻辑：R 语言数据挖掘之道

如上，scale 函数结构一般通过下画线分为三段（见表 3-1），以 scale 开头，中间指定图形属性的名称，这里是 size，如果是 X 轴就是 x，如果是 Y 轴就是 y，如果是颜色就是 colour 等，最后一段是指定相应的数据字段类型，这里映射为 size 的变量 Population 为连续型变量，所以用 continuous，如果变量是分类型变量，那就用 discrete。其实这些函数没必要全部记住，只需要记住结构和一些英文单词就可以了。如果你对一幅图的配件和语法结构都熟悉，那无论是试错，还是找代码参考都比较方便。

表 3-1

图形属性	英文关键字	根函数	衍生函数	备 注
X 轴	scale x axis	scale_x	scale_x_continuous	连续型变量的 X 轴
			scale_x_discrete	分类型变量的 X 轴
Y 轴	scale y axis	scale_y	scale_y_continuous	连续型变量的 Y 轴
			scale_y_discrete	分类型变量的 Y 轴
大小（面积或直径）	scale size radius	scale_radius	scale_radius	以圆直径为标尺
	scale size	scale_size	scale_size_area	以面积为大小变化标尺
			scale_size_continuous	连续型变量的大小变化标尺
			scale_size_discrete	分类型变量的大小变化标尺
颜色（形状及其边线）	scale colour	scale_colour	scale_color_brewer	使用 colorbrewer 提供的画板作为分类变量颜色变化标尺
			scale_color_gradient	固定色之间的平滑变化作为连续型变量的标尺
			scale_colour_manual	自定义颜色作为分类型变量颜色变化标尺
			scale_color_hue	调节色调变化作为分类型变量颜色变化标尺
填充色	scale colour fill	scale_fill	scale_fill_brewer	colorbrewer 提供的画板作为分类型变量颜色变化填充标尺
			scale_fill_gradient	固定色之间的平滑变化作为连续型变量的填充标尺
			scale_fill_manual	自定义颜色作为分类变量颜色变化填充标尺
			scale_fill_hue	调节色调变化作为分类型变量颜色变化填充标尺
			scale_fill_continuous	固定色之间的平滑变化作为连续型变量的填充标尺
透明度	scale Alpha	scale_alpha	scale_alpha_continuous	指定连续型变量的透明度变化标尺
			scale_alpha_discrete	指定分类型变量的透明度标尺
线类型	scale linetype	scale_linetype	scale_linetype_continuous	指定连续型变量的线类型标尺
			scale_linetype_discrete	指定分类型变量的线类型标尺
形状	scale shape	scale_shape	scale_shape_continuous	指定连续型变量的形状标尺
			scale_shape_discrete	指定连续型变量的形状标尺

表 3-1 列举了 ggplot2 中常见的标尺，这些标尺不仅在 ggplot2 中出现，其他任何绘图包也同样需要标尺，在其他语言中也需要标尺，只要存在变化，就存在衡量变化的标尺。太多名称很难记住，但是需要记住的是在可视化中，除了坐标轴以外，至少还存在形状、颜色、填充色、大小、粗细、线型等方面可以作为表达数据和规律的形式。另外需要记住一些相关的英文单词，这是为了抄代码。

其实大多数人大多数情况下不是在写代码，而是代码的搬运工：抄别人的或者抄自己的。前两天看到一个笑话，说编程其实很简单，一般分为 4 个步骤：① 在谷歌中用英文搜索问题；② 打开 Stack Overflow 连接；③ 复制粘贴代码（稍作修改）；④ 如果问题未解决请看第②条，如果问题解决了请看第①条。你会发现这 4 步本就是程序员思维的伪代码。

言归正传，我们距离一份印刷级别的图表还有很远的路。以上通过将数据映射到散点的大小、颜色了解了标尺的概念，下面需要对数据做一些简单的美化，不做修改实在有碍观瞻。首先需要自定义散点的颜色，使用 RGB 色系定义了一套双色颜色 cols1，对应 GDPmean 变量的两个分类级别，然后使用离散变量的标尺函数 scale\_colour\_manual 将 cols1 定义为散点颜色的标尺，如下所示。

- 自定义边线色

```
1 cols1 <- c(rgb(red = 0, green = 88, blue = 92, max = 255), rgb(red = 249,
green = 19, blue = 19, max = 255))
2 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
3   geom_point(aes(colour = GDPmean)) +
4   scale_size_continuous(range = c(1, 13)) +
5   scale_colour_manual(values = cols1)
```

很明显，这个颜色过于刺眼，丝毫不具有商务气质，但是先不要急着否定，这种问题可能不是颜色本身的问题，也许是使用方式不对。在 R 语言中，形状分为可填充形状和非可填充形状，可填充形状的颜色分为两种：边线颜色 colour 和填充色 fill；非可填充颜色仅有 colour 这一个属性。

那么问题来了，哪些形状是可填充形状呢？很多书里会列出一个表，我这里就不列了，列了大家也记不住，形状 21~25 号是可填充的，大家通过下面的代码查看 R 中可以使用的形状，快速略过，这里我使用了基础绘图包。

- R 中的形状 (Symbols)

```
1 i <- 0:25
2 plot(i, pch = i, bg = "blue")
```

上述代码中，其中参数 pch 是“plotting ‘character’”的缩写，用于指定使用的形状编号，bg 是 background 缩写，用于指定背景色。其实我觉得书中能够用代码实现的图就不用做成插图了，大家都是技术员，何必浪费纸张呢，但话又说回来，没有漂亮的插图又怎么吸引人呢？

学习了 R 中的形状一方面丰富了我们用于展示数据的形状库，另一方面也拓展了我们对形状颜色属性的理解。那么有了这个基础，就可以对上面的散点图做进一步改造了，cols1 在整个圆点上使用面积太大，看起来刺眼，我们不妨将其设为散点的边线颜色，另外再自定义一组颜色作为散点的填充色，为了配合这次行动需要将散点的形状编号设为 21 号，即具有边线颜色和填充色双重颜色属性的圆形符号，如下所示。

- 自定义填充色

```
1 cols <- c(rgb(red = 167, green = 251, blue = 255, max = 255), rgb(red =
254, green = 218, blue = 218, max = 255))
2 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
3   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21) +
4   scale_size_continuous(range = c(1, 13)) +
5   scale_colour_manual(values = cols1) +
6   scale_fill_manual(values = cols)
```

效果如图 3-32 所示。

上面代码中，参数 fill 指定了映射变量（指定映射的都在 aes 函数中），参数 shape 指定了形状编号。scale\_fill\_manual 函数指定了填充色标尺，因为 GDPmean 是分类变量，所以标尺均用的是 manual 后缀。到这里大家应该发现我们指定色彩的 aes 参数全部使用的是局部控制，即仅仅控制这一层的散点颜色，原因是我并不想使用全局控制直接把数据按照 GDPmean 变量分为两个部分。

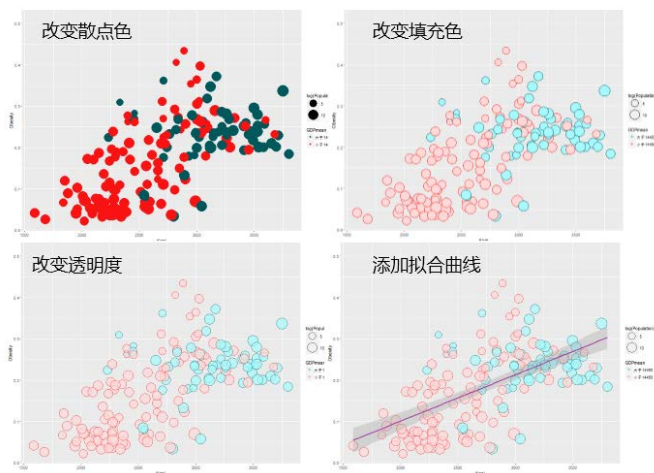


图 3-32

如图 3-32 所示，世界一下子清新了许多。但是新的问题出现了，很多散点重合了，处理形状重合的方法我们已经在 3.1.4 节讲过了：改变透明度和随机移动。后者可以通过 jitter 函数实现，但是我不建议在与数据映射相关的地方使用 jitter，因为一旦移动便不是原来的数据了，违反了数据可视化准确性表达的原则，如果是标签类的重合则可使用 jitter 解决（比如绘制地图时），毕竟标签不是数据的映射，只是为了方便解读。这里使用透明度（alpha）缓解一下重合现象，如下所示。

- 修改透明度

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols)
```



有读者问，能不能将 `shape` 和 `alpha` 通通移到 `aes` 参数里去，这样虽然不会报错，但是意义和效果不一样，放在里面表示从数据到形状的映射，放在外面表示属性使用的标尺，前者会在图例中多出个不伦不类的东西。所以在 `aes` 里尽量放置映射关系，比如“`alpha = 某变量`”，指定单个属性值还是放在外面作为 `geom` 函数的参数比较合适。

重提一下，我们做这幅图研究的目的是揭示肥胖人口比例和饮食热量的关系，要看一下肥胖人口比例和饮食热量相关性有多大，是否像《信号与噪声》中所说的那样基本不存在相关性。可以检验 `Obesity` 和 `Kcal` 之间的相关性，或者进一步建立线性模型分析一下，如下所示。

- 检验相关性

```
1 cor.test(Obesity$Obesity, Obesity$Kcal)
2 linemodel1 <- lm(Obesity ~ Kcal, data = Obesity)
3 summary(linemodel1)
4 linemodel2 <- lm(Obesity ~ Kcal - 1, data = Obesity)
5 summary(linemodel2)
```

通过相关性检验我们发现肥胖人口比例和饮食热量之间存在非常显著的相关性，相关系数接近 0.6。进一步建模建立线性模型发现线性模型的系数经检验均达到极显著水平，但是这里出现了一个有趣的现象：拥有常数项的线性模型 `R` 平方仅为 0.35，而去掉常数项（在模型方程后面-1）的线性模型 `R` 平方高达 0.85。

那么问题来了，首先，要不要加常数项？这个是由模型的现实意义决定的，按照模型 1 的意义解释：当一个国家的饮食热量为 0 时，这个国家的肥胖人口比例出现负值，但这样的解释和现实意义是不符的，如果一个国家的居民饮食热量为零，那这个国家的肥胖比例最少也就是 0 吧，无论如何不会出现负值。尽管统计意义上说无常数项模型是有常数项模型的特殊形式，有常数项模型更加稳健，但是我认为（非学术）在项目应用中还是尽量按照现实意义解释为妙。在《数字是靠不住的》一书中就列举了 *Nature*、*Science* 发表的大量违反现实意义的模型。

#### 在什么情况下要添加常数项

（1）如果现实情况下自变量为零时，因变量为正，建模后常数项为负，可以考虑去除常数项，反之添加（特别是在检验显著的情况下）。

（2）如果现实情况下自变量为零时，因变量为负，建模后常数项为正，可以考虑去除常数项，反之添加（特别是在检验显著的情况下）。

（3）如果现实情况下自变量为零时，因变量为零，可以考虑去除常数项。

那么为什么会出现这种现象呢？主要是因为 `Obesity` 和 `Kcal` 两个变量的数字相差好几个数量级，所以碰到这种情况应该先对数据进行标准化之后再做统计建模，为节省篇幅这里不再验证。

经过上面的代码，清楚了肥胖人口比例和饮食热量之间存在非常显著的正相关这一事实，那么就有必要把 `Obesity` 和 `Kcal` 之间的这种规律反映到气泡图上，在图上添加拟合曲线和置信区间，图表蕴含的信息就更加丰富了，同时也向受众传达了数据挖掘的价值。

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
```

```
4 scale_colour_manual(values = cols1) +
5 scale_fill_manual(values = cols) +
6 stat_smooth(method=lm, alpha = 0.3, size = 1, colour= rgb(red = 162, green
= 57, blue = 165, max = 255))
```

上述代码中，stat\_smooth 函数用于添加拟合曲线和置信区间，我们使用了几个参数：method 用于设定拟合方法，这里使用线性拟合设为 lm；alpha 和 size 分别用于设定透明度和线的粗细，显然不能让它们遮挡了气泡；colour 用于设定线的颜色。当然 stat\_smooth 还有很多其他的参数，比如去除置信区间的阴影(se = FALSE)、设置置信度(level)等，这里的统计变换即为绘制 Obesity 变量对 Kcal 的线性拟合曲线，并绘出预测值的 95% 的置信区间。大家可以看到，我们在 stat\_smooth 中并没有指定数据和变量，也就是说如果不特殊指定，ggplot2 默认将自动从全局设置(ggplot 函数)中继承。

如果就此打住，那就和其他教程一样了，因为这里接触了 ggplot2 包的另一个重要的概念，或者称之为函数系：统计变换(stat\_)。ggplot2 的思想认为每一个几何对象都有一个默认的统计变换，并且每一个统计变换也都有一个默认的几何对象，但是我们暂且不提这种广义的统计变换，主要说一下 ggplot2 独立出来的统计变换，这往往和统计学的具体概念相关，比如拟合曲线、置信区间、频次直方图等，ggplot2 把这类常见的概念独立出来使用简便的函数可视化数据中隐藏的具有统计意义的知识，这一点确实是优于其他绘图包的地方。下面的代码挑选紧要的统计变换做些说明。

- 统计变换(stat)

```
#频率(频度)柱状图
##对变量按因子计数，将频数绘制为柱图
1 ex <- ggplot(Obesity, aes(GDPmean)) + geom_bar(stat="count")
# stat 是默认的统计变换，不添加 stat 参数也可以
2 ex
#绘制步长频率图
##连续变量分组，统计每一段的观测值的个数，将各组频数绘制柱图，便于观察数据分布
3 ex <- ggplot(Obesity, aes(GDPPPP)) + geom_bar(stat="bin")#不能调节分段的
#步长和步数
4 ex
##调节步长和步数
5 ex <- ggplot(Obesity, aes(GDPPPP)) + stat_bin(binwidth = 1000)#步长设置
#为 1000
6 ex
#绘制密度函数曲线
##通过频度图可以模拟出密度函数曲线
7 ex <- ggplot(Obesity, aes(GDPPPP)) + geom_density()
8 ex
#正态 QQ 图
##用于直观验证一组数据是否来自某个分布，或者验证某两组数据是否来自同一(族)分布
9 ex <- ggplot(Obesity, aes(sample = Kcal)) + stat_qq(na.rm =TRUE)
10 ex
```

```
#累计百分比曲线
##ecdf 为 Empirical Cumulative Density Function 缩写, 模型评价时会再次提及
11 ex <- ggplot(Obesity, aes(Kcal)) + stat_ecdf(geom = "step")
12 ex
#绘制函数
13 ex <- ggplot(Obesity, aes(Kcal)) + stat_ecdf(geom = "step")
14 ex + stat_function(fun = sin, colour = "red")#函数也可以是用户自定义的函数, 如下
##test <- function(x) {x ^ 2 + x + 20}
```

上面代码是连续型变量分组统计频率研究数据分布的频率图, 这就有一个 binwidth 的问题, 我译为步长或组距, 做出来的频率图可以初步反映数据的分布特征, 比如可以看出哪个收入段内的居民人数最多。ggplot2 提供了频数 (stat\_count)、分组频数 (stat\_bin)、密度曲线 (stat\_density)、正态 QQ 图 (stat\_qq)、箱线图 (stat\_boxplot)、拟合曲线 (stat\_smooth)、摘要探索 (stat\_summary)、函数模拟 (stat\_function)、累计百分比曲线 (stat\_ecdf) 等多种变换方式。就我而言, 其中最常用的只有拟合曲线, 其实这个功能和函数模拟 (stat\_function) 作用相似, 如果知道了函数方程, 通过自定义函数同样可以绘制拟合曲线。

这么多函数实在记不住, 而且有些函数在后来的版本中已经不以 stat 开头, 而是以 geom 开头了。但这不是重要的事情, 也不是拒绝学习的理由和借口。

把业务问题转化为统计语言或数据语言的能力才是数据分析师的核心能力。

其实重要的还是应用, 把业务问题转化为统计语言, 比如想知道全球人口分布是不是符合二八定律, 是不是全球 20% 的国家占据了全球 80% 的人口 (见图 3-33)。同样, 研究一家网店的顾客贡献度是不是符合二八定律, 从而找出优质客户, 这个问题就是数据的分布问题(对长尾效应图的变形), 可以用客户贡献度百分比累计图非常恰当地说明。

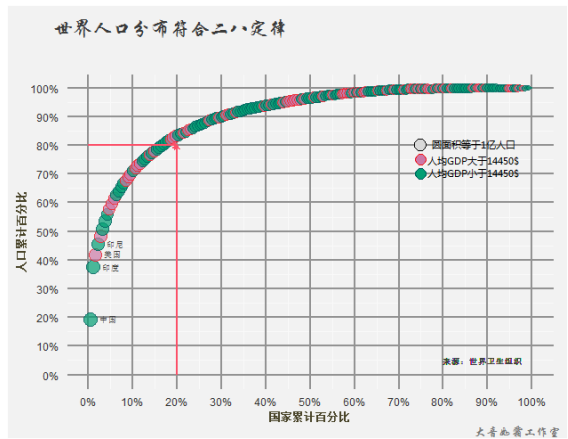


图 3-33

从图 3-33 中可以看出, 世界人口分布基本上符合二八定律。有读者说只想绘制一幅条形图或柱

形图(按照具体的值绘制条形),而不是频率图,怎么办?将 `geom_bar` 的统计变换设为 `identity` 即可,如下代码所示。至于条形图囊括的三种形式,我们先挖个“坑”,将在后面的基础图形里详细讲解。

- 绘制柱形图

```
1 df <- data.frame(trt = c("a", "b", "c"), outcome = c(2.3, 1.9, 3.2))
2 ggplot(df, aes(trt, outcome)) +
3   geom_bar(stat = "identity")
```

但是有一个秘密不得不说一下,即统计变换里的计数变换 `geom_bar(stat="count")`或分组计数变换 `geom_bar(stat="bin")`,当数据量很大时,显然在绘图时计数效率会非常慢,特别是图表嵌在交互的网页内时,这时 `geom_bar(stat = "identity")`就派上了用场,可以事先完成计数做成表格,然后用户点击交互时直接根据表格绘制频率图即可,从而减少了反映的时间消耗。

即使在其他绘图工具中,同样要通过各种方式将数据、规律做恰当地统计变换展示给受众。

在以后的图表中,统计变换肯定是不可或缺的,原因很简单,统计变换不仅把数据中挖掘的知识展示在图表中,辅助受众理解数据的真实含义,同时很多统计变换自身就是挖掘出来的趋势或规律,直接可以用于决策。毫无疑问,数据挖掘如若向受众展示自己最诱人最成熟的部分,统计变换可视化是不二之选,另外统计变换可能是一般 BI 软件的短板,而恰恰又是数据挖掘暂时为人称道的地方(很多方法和应用不成熟,与人们想象中相差甚远)。

到这里,我们学习了 `ggplot2` 的图层(`geom`)、映射(`aes`)、标尺(`scale`)、统计变换(`stat`)等概念,但是还是没有做出一幅像模像样的图表,当然你也可以使用默认设置完成任务。

在职场中,任何一份成果都是你专业水平和用心程度的表现,客户看到,上司也会看到。

虽然经过上面的代码,气泡图有点模样了,重要的元素都已经出现在图表上,但是若要它成为印刷级的作品,还需要我们继续灌注心血和学识。有时候,完成一份自己满意的作品,不仅是得到别人的认可和赞誉,更重要的是内心收获一种知识的愉悦。

书接上文,代码当然也紧接着上文。参看已经绘制的气泡图,不难发现,由于图表是静态图表,不具有互动功能,若是 D3.js 类型的图表,当用户的鼠标指向某一个对象时会显示对象的一些属性,但是静态图表就像户外的静态广告一样,画面上有啥你就看到啥,不具有互动功能,所以有必要在静态图表中有选择性地标出一些内容,比如标签,方便受众认知。

要把一些特殊的国家标识出来,首先是中国,因为看到这幅图的人大部分是中国人,自然要把中国标识出来,就好像你一定要把客户的产品在分析图表中充分强调一样;其次要根据与业务或主题相关的标准筛选出来一些特殊点打标签。

### 特殊点筛选

- (1) 热量摄入比较多或者肥胖比例比较高的国家;
- (2) 热量摄入中等以上且肥胖比例比较低的国家;

## (3) 热量摄入不足的国家。

第(1)点换句话说就是“吃货”比较多,或者胖子比较多的国家。提醒大家的是,这是“或”的关系,不是“且”的关系。不要轻视“或”与“且”这些小细节,在大型项目中这些关系搞错一般只有在上线被人“打脸”后才能发现。条件a转化为数据语言,即是将热量摄入超过3600Kcal或者肥胖比例超过35%的国家筛选出来。如下代码所示。这里有一个常识需要了解一下:正常人一般每天至少需要消耗2000Kcal的热量。

- 标签筛选条件 a

```
#全局设定标签映射
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols) +
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0)
#局部设定标签映射
8 ex <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population))) +
9   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha =
0.7) +
10  scale_size_continuous(range = c(1, 13)) +
11  scale_colour_manual(values = cols1) +
12  scale_fill_manual(values = cols) +
13  stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
14  geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), aes(label
= Country), size = 3.0, colour = "gray25", hjust = 0, vjust = 0)
```

首先代码做了两处改变,在全局控制函数(当然不一定非得在全局里定义,参见局部代码)ggplot的参数aes里将变量Country映射为标签(label),这里只是定义了映射,并没有绘制图层,所以我们使用geom\_text函数添加了图层。如果不加筛选,则表示将所有的国家名称作为标签,会出现满屏标签的情况(geom\_text后面括号里的内容抹掉试试),这显然不是广大读者喜闻乐见的东西,所以需要筛选分组,subset函数根据条件提取数据对象的子集,“|”表示“或”的关系。随后设置了标签的大小(size)和颜色(colour),并通过hjust和vjust参数对标签对齐方式(通常误理解为位置)做了适当调整,标签的位置已经由对应的坐标值标定,所谓调整就是相对于坐标值的调整。

#### 对齐方式调整参数hjust和vjust

① hjust: 水平对齐调整(horizontal justification),值等于0.5时表示居中,即对象(标签)的中点和坐标点对应,0~0.5表示左侧对齐,即对象(标签)的左边界和坐标点对应,看起来像向右调

整，0.5~1 表示右侧对齐，即对象（标签）的右边界和坐标点对应。

② `vjust`：上下对齐调整（vertical justification），值等于 0.5 时表示居中，即对象（标签）的中点和坐标点对应，0~0.5 表示下侧对齐，即对象（标签）的下边界和坐标点对应，看起来像向上调整，0.5~1 表示上侧对齐，即对象（标签）的上边界和坐标点对应。

确切说，`hjust` 和 `vjust` 参数调整的并不是标签的位置，而是标签的对齐方式，当然其附带效应看起来像调整了位置。如若真的想调整标签位置，只能通过改变其坐标的方式进行，这里分别对标签进行了左对齐和下对齐。大家注意，这两个参数在所有绘图包中都是通用的名称，属于基础参数。

条件 b 转化成数据语言，即肥胖比例低于 10% 且热量摄入在 2600Kcal 以上的国家，换句话说就是那些吃不胖的人群待的地方，这些值需要根据统计特征或具体的业务设置。

- 标签筛选条件 b

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols) +
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
8   geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
3.0, colour = "gray25", hjust = 0, vjust = 0)
```

继续使用全局映射，其好处就是没有必要在每一个图层都指定一次标签与数据的映射关系。对数据进行了一次分组筛选（`subset`），但是这次的关系“且”，使用 `&` 函数表达，它表示肥胖比例低于 10% 且热量摄入在 2600Kcal 以上的国家。

条件 c 转化成数据语言就是筛选出热量摄入低于 2000Kcal 的国家。

- 标签筛选条件 c

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols) +
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
8   geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
9   geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour = "gray25",
```

```
hjust=0, vjust=0)
```

重要散点添加标签的任务已经基本完成了，好在标签重叠的现象并不严重，至于处理标签重叠的方法，后面再讲，绝不是简单地通过 jitter 函数就能解决的。要建立一个共识：无论是广告还是图表，静态的水平图表要求比动态图表高，很多图表的默认设置就是为了适应一般规律或者动态互动而设计的。就图例而言，很难通过已有的函数调整将其融入图表的风格之中，无法达到静态图表的要求，所以这里索性不要默认的图例了（不建议动态和交互图表系统使用该策略）。

静态图表：通过一幅静态画面讲述故事。

动态图表：通过时间序列式的逻辑画面或者互动效果讲述故事。

不使用默认的图例并不代表不需要图例，图例是受众理解图表的窗口，不能轻易移除，所以要重新绘制一个自定义的图例。从本散点图的映射关系分析需要绘制这些辅助理解的图例：① 大小；② 颜色。首先，要找一个合适的地方摆放新图例，一般图例默认放置在右边界、上边界或下边界，这显然是为了统一而做的权宜之计，既然要自定义图例，肯定不能再墨守成规，最合适的地方就是放在图表中大量的空白区域（本图在左上方）。

scale\_size 默认情况下以面积变化衡量变量变化的尺度，如果以直径衡量，则需要 scale\_radius 函数进行设置。一定要注意，我们到底将变量映射为了几何对象的哪个属性，之前将 log(Population) 设置为 size 的映射，所以需要有一个图例告诉受众一亿人口的圆应该有多大，Population 变量的单位是千人，只需要知道 log(100000) 的大小即可，ggplot2 将自动将 log(100000) 转化相应面积的圆，代码如下所示。

- 添加大小图例

```
1 if (!suppressWarnings(require("extrafont"))){
2   install.packages("extrafont")
3   require("extrafont")
4 }
5 #font_import()
6 loadfonts(device="win")
7 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
8   label = Country)) +
9   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
10  scale_size_continuous(range = c(1, 13)) +
11  scale_colour_manual(values = cols1) +
12  scale_fill_manual(values = cols) +
13  stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
14    green = 57, blue = 165, max = 255)) +
15  geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
16    3.0, colour = "gray25", hjust = 0, vjust = 0) +
17  geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
18    3.0, colour = "gray25", hjust = 0, vjust = 0) +
19  geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour =
```

```
"gray25", hjust = 0, vjust = 0) +  
  15 geom_point(aes(x = 1725, y = 0.45, size = log(100000)), shape = 21, colour  
= rgb(red = 7, green = 7, blue = 7, max = 255), fill = rgb(red = 217, green =  
217, blue = 217, max = 255)) +  
  16 annotate("text", label = "圆面积等于 1 亿人", x = 2000, y = 0.45, size =  
4, colour = "black", family = 'Microsoft YaHei')
```

在上述代码中，使用 `geom_point` 函数在横坐标为 1725，纵坐标为 0.45 的地方添加一个面积为  $\log(100000)$  的圆，然后设置了圆的边线颜色和填充色；另外需要添加一个标签对圆面积进行说明，使用 `annotate` 函数添加文字。这里需要介绍一下 `annotate` 函数，它与 `geom` 的作用一样，用于添加图层，唯一的区别是 `annotate` 不要求数据源是 `data.frame`，只要是向量即可，一些临时的或者零散的数据可能并没有包含在图表的数据框中，而是需要临时添加。`annotate` 添加的几何对象包括文字（`text`）、面（`rect`）、线段（`segment`）等，第 1 个参数用于设置添加对象的类型，这里是文字，`label` 用于设置添加文字的内容，可以是一个向量，也就是说可以添加多个标签。参数 `x`、`y` 用于指定标签的位置，它们也可以向量化，最后指定了文字的颜色和字体类型。

上面还加载了一个包 `extrafont`，它的作用如下所示。

- 怎么使用 Windows 系统的自带字体

```
1 if (!suppressWarnings(require("extrafont"))){  
2   install.packages("extrafont")  
3   require("extrafont")  
4 }  
5 font_import()#这一步用完了就注释掉  
6 loadfonts(device="win")
```

R 的默认字体比较少，所以不得不想办法使用系统自带的字体，特别是在 Windows 系统下。`extrafont` 完美地解决了这个问题，首先要对字体进行一次引入，这次引入是一次性的，以后再次使用就不需要重复引入了，毕竟引入花费的时间还是比较长的。之后再使用外来字体时需要加载一次包和字体 `loadfonts`，`family` 参数指定相应的字体即可，例如我们指定了微软雅黑（Microsoft YaHei）。

当一个人开始发现问题时，说明他具备了专业的视角，当一个人推敲问题并试图改变时，他才真正具备了专业的技能。

如果仅仅添加一个圆，受众无法推算出比图例大的圆代表多少人口，小的又是多少人口，所以图例必须讲明变化的尺度，还需要添加一个大小为  $\log(10000)$  的圆，用于标定从一千万人口到一亿人口变化的尺度，如下所示。

```
1 ex <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),  
label = Country)) +  
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +  
3   scale_size_continuous(range = c(1, 13)) +  
4   scale_colour_manual(values = cols1) +  
5   scale_fill_manual(values = cols) +  
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
```



```

green = 57, blue = 165, max = 255)) +
  7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
  8   geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
  9   geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour =
"gray25", hjust = 0, vjust = 0) +
  10  geom_point(aes(x = 1725, y = 0.45, size = log(100000)), shape = 21, colour
= rgb(red = 7, green = 7, blue = 7, max = 255), fill = rgb(red = 217, green =
217, blue = 217, max = 255)) +
  11  annotate("text", label = "圆面积等于1亿人", x = 2000, y = 0.45, size =
4, colour = "black", family = 'Microsoft YaHei') +
  12  geom_point(aes(x = 1725, y = 0.42, size = log(10000)), shape = 21, colour
= rgb(red = 7, green = 7, blue = 7, max = 255), fill = rgb(red = 217, green =
217, blue = 217, max = 255)) +
  13  annotate("text", label = "圆面积等于1千万人", x = 2020, y = 0.42, size
= 4, colour = "black", family = 'Microsoft YaHei')

```

以上出现了两次完全相同的操作，`annotate` 函数的向量作用，完全可以把四行代码缩减为两行，如下所示。

- `annotate` 函数的向量作用

```

1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha =
0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols) +
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
8   geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
9   geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour =
"gray25", hjust = 0, vjust = 0) +
10  annotate("point", x = c(1725, 1725), y = c(0.45, 0.42), size =
c(log(100000), log(10000)), shape = 21, colour = rgb(red = 7, green = 7, blue
= 7, max = 255), fill = rgb(red = 217, green = 217, blue = 217, max = 255)) +

11  annotate("text", label = c("圆面积等于1亿人", "圆面积等于1千万人"), x =
c(2000, 2020), y = c(0.45, 0.42), size = 4, colour = "black", family = 'Microsoft
YaHei')

```

不难发现，`annotate` 函数除了第1个参数用于指定几何对象，不能以向量的方式使用外，其他参

数都可以用向量的方式使用。按照这个思路，上面添加国家标签的代码也可以使用一行代码完成，有想法的读者可以尝试一下。

完成了解释大小的图例，下面就是绘制解释颜色的图例了，如下所示。

- 绘制解释颜色的图例

```
1 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
2   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
3   scale_size_continuous(range = c(1, 13)) +
4   scale_colour_manual(values = cols1) +
5   scale_fill_manual(values = cols) +
6   stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
7   geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
8   geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size = 3.0,
colour = "gray25", hjust = 0, vjust = 0) +
9   geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour =
"gray25", hjust = 0, vjust = 0) +
10  annotate("point", x = c(1725, 1725), y = c(0.45, 0.42), size =
c(log(100000), log(10000)), shape = 21, colour = rgb(red = 7, green = 7, blue
= 7, max = 255), fill = rgb(red = 217, green = 217, blue = 217, max = 255)) +
11  annotate("text", label = c("圆面积等于 1 亿人", "圆面积等于 1 千万人"), x =
c(2000, 2020), y = c(0.45, 0.42), size = 4, colour = "black", family = 'Microsoft
YaHei') +
12  annotate("point", x = c(1725, 1725), y = c(0.39, 0.365), size = c(8,
8), shape = 21, colour = c(rgb(red = 0, green = 88, blue = 92, max = 255), rgb(red
= 249, green = 19, blue = 19, max = 255)), fill = c(rgb(red = 167, green = 251,
blue = 255, max = 255), rgb(red = 254, green = 218, blue = 218, max = 255))) +
13  annotate("text", label = c("人均 GDP 大于 14450$", "人均 GDP 小于 14450$"),
x = c(2025, 2025), y = c(0.39, 0.365), size = 4, colour = "black", family =
'Microsoft YaHei')
```

上述代码中再次使用 `annotate` 函数添加了两个圆，横坐标均为 1725，纵坐标分别为 0.39、0.365，大小都是 8，使用了和默认图例相同的边线颜色和填充色，而后又添加了两个标签用于解释图例。是不是可以进一步缩减代码，和之前的解释大小的图例一起，一次添加 4 个圆或者一次添加四个标签呢？或者使用 `geom_point` 分 4 次添加圆和标签呢？这里不再给出代码，想深入了解可以自行尝试。另外，以上关于坐标的数字大家可以根据自己的画面大小进行调整以期达到最佳效果。

既然有了自己绘制的图例，那原有的图例就可以不要了，只需在 `theme` 函数中将图例的位置设置为空就可以了，代码如下所示。

- 去除图例

```
p <- p + theme(legend.position = "")
```

主题 `theme` 函数可以称得上“子孙满堂”了，这个留在后面详细说明，先讲一下坐标轴，大家

想一想坐标轴有哪些元素？稍加观察不难发现，坐标轴至少存在以下几点有待修改：① 坐标轴名称应该为中文；② Y 轴的刻度标签应改为百分比计数；③ X 轴右边界有待拓宽，奥地利已经在图的边缘了，代码如下所示。

- 修改坐标轴标签、百分号，拓宽 X 轴右边界

```
1 if (!suppressWarnings(require("scales"))){
2   install.packages("scales")
3   require("scales")
4 }
5 p <- ggplot(Obesity, aes(x = Kcal, y = Obesity, size = log(Population),
label = Country)) +
6   geom_point(aes(colour = GDPmean, fill = GDPmean), shape = 21, alpha = 0.7) +
7   scale_size_continuous(range = c(1, 13)) +
8   scale_colour_manual(values = cols1) +
9   scale_fill_manual(values = cols) +
10  stat_smooth(method = lm, alpha = 0.3, size = 1, colour = rgb(red = 162,
green = 57, blue = 165, max = 255)) +
11  geom_text(data = subset(Obesity, Obesity > 0.35 | Kcal > 3700), size =
= 3.0, colour = "gray25", hjust = 0, vjust = 0) +
12  geom_text(data = subset(Obesity, Obesity < 0.1 & Kcal > 2600), size =
3.0, colour = "gray25", hjust = 0, vjust = 0) +
13  geom_text(data = subset(Obesity, Kcal < 2000), size = 3.0, colour = "gray25",
hjust = 0, vjust = 0) +
14  annotate("point", x = c(1725, 1725), y = c(0.45, 0.42), size =
c(log(100000), log(10000)), shape = 21, colour = rgb(red = 7, green = 7, blue
= 7, max = 255), fill = rgb(red = 217, green = 217, blue = 217, max = 255)) +
15  annotate("text", label = c("圆面积等于 1 亿人", "圆面积等于 1 千万人"), x =
c(2000, 2020), y = c(0.45, 0.42), size = 4, colour = "black", family = 'Microsoft
YaHei') +
16  annotate("point", x = c(1725, 1725), y = c(0.39, 0.365), size = c(8,
8), shape = 21, colour = c(rgb(red = 0, green = 88, blue = 92, max = 255), rgb(red
= 249, green = 19, blue = 19, max = 255)), fill = c(rgb(red = 167, green = 251,
blue = 255, max = 255), rgb(red = 254, green = 218, blue = 218, max = 255))) +
17  annotate("text", label = c("人均 GDP 大于 14450$", "人均 GDP 小于 14450$"),
x = c(2025, 2025), y = c(0.39, 0.365), size = 4, colour = "black", family =
'Microsoft YaHei') +
18  ylab("肥胖比例 (BMI >= 30 kg/m2)") +
19  xlab("食物能量消耗 (人每天 kcal)") +
20  scale_x_continuous(limits = c(1500, 4100)) +
21  scale_y_continuous(labels = percent)
```

上述代码中，ylab 和 xlab 函数分别用于修改 Y 轴名称和 X 轴名称，请注意这里修改的是坐标轴名称 (axis.title)，而不是坐标轴刻度标签 (axis.text)，尽管 ylab 里有一个 label 的缩写，使用 scale\_x\_continuous 更改了坐标轴的起始位置，前面已经讲过不再赘述。

上面加载了一个新包 `scales`，也是 Hadley 开发的，是 `ggplot2` 的附属包，顾名思义，这个包是用来调整标尺的，这里使用它的 `percent` 函数将  $Y$  轴的刻度标签转化为百分比格式。

数据来源是权威的象征，所以我们还需要在图上恰当的位置标注数据来源，和之前一样，使用 `geom_text` 添加一个文字层就可以了。通过 `aes` 参数设置文字位置，这里设置在右下角，然后设定标签内容为“来源：世界卫生组织”，最后指定颜色、水平对齐方式和字体大小，如下所示。

- 添加标题

```
# 自编添加主副标题函数
1 title_with_subtitle = function(title, subtitle = "") {
2   ggtitle(bquote(atop(.(title), atop(.(subtitle)))))
3 }
4 p <- p + geom_text(aes(3600, .01), label="来源：世界卫生组织", colour="gray25",
hjust=0, size=4.0) +
5   title_with_subtitle("为什么我们生活的世界总有些人是瘦子", "各国食物能量消耗与
肥胖比例反馈“吃得少”")
6 a <- 3
7 b <- 2
8 bquote(y == sqrt(.(a), .(b)))
```

图表标题往往具有画龙点睛的作用，所以拟定标题时一定要三思。这里基于 `ggplot2` 用于添加标题的 `ggtitle` 函数，自编了一个添加主副标题的函数 `title_with_subtitle`，函数包括主副标题两个参数，我们先从最里层解析这个函数：`subtitle` 用于引用对象；参数 `atop` 表示将一个对象放在另一个对象之上的意思，两个 `atop` 嵌套在一起只是为了保证主副标题分开处理，第 2 个 `atop` 让副标题略小于主标题；`bquote` 的作用是反引用，即你可以在标题中引用具体的 R 对象（关于引用和表达式 `expression` 请参看基础章节，或如上例），这里就保证如果你要引用 R 中的对象作为标题的一部分也是可行的。其实，有读者会说，为什么费这么大劲，直接添加标题，在主副标题间添加个换行符不就成了，是成了，但是你能分别控制字体吗？你能分别控制大小吗？

万里长征路走到这里，图上的元素基本上各居其位了，但是看起来仍然不尽如人意。原因很简单，很多与风格相关的元素还没有加进来，比如字体、背景、主次网格线、坐标轴刻度线等，`ggplot2` 里统称为主题元素（`theme`）。下面开始塑造个人或公司的风格吧？

- 添加空白主题图层

```
p <- p + theme_bw(18)
```

上述代码中，使用 `theme_bw` 函数用于添加一个只包含灰色网格线的白色背景图层，它有两个参数：`base_size` 设置默认文字大小；`base_family` 用于设置默认字体，属于全局控制。

所有非数据映射类型的元素都由 `theme` 控制，包括字体、字体大小、坐标轴刻度线、网格线、背景色、图表边界、图表边界线、图例等，即所有体现绘图风格的非数据元素。

`ggplot2` 包里的另一个重要概念 `theme` 就此“粉墨登场”，这里要用到大量图表结构的知识，不懂的读者请参看前面的章节，首先要设定图表边界（`plot.margin`），`unit` 函数用于设置图表的下、左、上、右边界区域宽度。为什么要设置图表边界？回顾一下就知道，坐标轴名称、图表标题等元素均

位于图表边界内，需要预留足够的空间。Unit 的第 2 个参数设置数值的单位，它有两种标准：inches 和 lines，前者很容易理解，就是英寸的意思，后者就难以理解了，为左边界预留 1 lines 空间是什么意思？这里 lines 表示一行字体高度（预设的或正在使用的字体）空间大小，是一个相对变量单位。

- 设定图表边界、图表区背景、绘图区背景

```
1 p <- p + theme(plot.margin = unit(c(1, 1, 1.25, 0.5), "lines"),
2               plot.background = element_rect(fill = rgb(red = 242, green = 242, blue = 242, max = 255)),
3               panel.background = element_rect(fill = rgb(red = 242, green = 242, blue = 242, max = 255)),
4               panel.border = element_rect(colour = rgb(red = 242, green = 242, blue = 242, max = 255)))
```

上述代码的中间两行代码设置了图表区和绘图区的填充色，统一为灰色。需要注意的是图表区和绘图区是两个不同的区域，尽管绘图区在图表区内，它们的颜色仍然要分别设置，并不是指定一下 background 就可以的，这一点在 Excel 的图表里也是同样的概念。

最后一行代码把绘图区的边界颜色设置为与背景相同的灰色，以使绘图区和图表区融为一体，而不是分隔的区域。

其实 theme 里的函数也有规律，一般分为“=”前后两个部分，等号前面是对象（图表元素）的名称，等号后面是待设定元素的类型命名的函数，总共有 5 种：单位 unit、平面 element\_rect、线 element\_line、移除元素 element\_blank、文字 element\_text 和“”空值。后面函数的选择和前面对象的属性一一对应，比如上面设置的背景都是平面的属性，所以均用了 element\_rect，几乎所有的对象都可以通过 element\_blank 设置为空白。

- 设定图表标题、网格线

```
1 p <- p + theme(plot.title = element_text(size = rel(1.2), family = 'STXingkai', face = 'bold', hjust = 0.05, vjust = 1, colour = rgb(red = 51, green = 63, blue = 79, max = 255)),
2               panel.grid.major=element_line(colour=rgb(red = 146, green = 146, blue = 146, max = 255),size=.75))
```

上述代码中图表标题就是 plot.title，很明显元素类型应该是 element\_text。之所以难记是因为对这个东西了解得不够深入，如果知道图的元器件的名称，记这些琐碎会方便很多；设置字体的大小，这里使用了 rel 函数，它设置的是相对大小（relative size），相对于谁？相对于前面 theme\_bw() 设置的默认字体大小 18，为其 1.2 倍；然后设置了字体、粗细、水平对齐方式、上下对齐方式和字体颜色。

第 2 行代码设置了主网格线的属性，主网格线是 panel.grid.major，次网格线自然是 panel.grid.minor，线一般就有粗细、颜色和方向三个属性，上面设置了主网格线的前两个属性。

- 设定坐标轴相关属性

```
1 p <- p + theme(axis.ticks = element_blank(),
2               axis.text.x = element_text(colour="grey20", size=12),
3               axis.text.y = element_text(colour="grey20",size=12),
```

```
4      axis.title.y = element_text(size = 11, colour = rgb(red = 74,
green = 69, blue = 42, max = 255), face = "bold", vjust = .5),
5      axis.title.x = element_text(size = 11, colour = rgb(red = 74,
green = 69, blue = 42, max = 255), face = "bold", vjust = .5),
6      legend.position = "")
```

上述代码中，由于已经有了漂亮的网格线，坐标轴刻度线就暂无存在的价值了，首先将其设置为 `element_blank`；然后分别设置了 X 轴、Y 轴刻度标签的颜色和大小；显然坐标轴的名称（title）也需要修改一下，设置了字体大小、颜色，同时为了醒目进行了加粗和对齐方式居中处理；最后空值移除了默认图例。

完成的图表作品要输出保存，下面就将作品保存为 PNG 格式的图片。

- 加载 LOGO

```
1 if (!suppressWarnings(require("grid"))) {
2   install.packages("grid")
3   require("grid")
4 }
5 add_credits = function(fontsize = 12) {
6   grid.text(label = "大音如霜工作室",
7             x = 0.99,
8             y = 0.02,
9             just = "right",
10            gp = gpar(fontsize = fontsize, col = "#777777", fontfamily =
'STXingkai'))
11 }
```

可能看到上面的输出图片的代码，有些读者又要问：输出一张图片有那么复杂吗？我觉得也没有，但是把问题搞复杂不是我们的长项。在输出图片的同时需要给图片添加公司的 LOGO 和标记，报告中图表蒙上小巧的公司标志是一件格调很高的事情，上面就是一个自编的添加 LOGO 和标记的函数，只是看起来复杂了而已，但一切为了专业，专业就是绘图之道。

执行添加标记的函数 `add_credits` 需要加载 `grid` 包，这是一个基础包，提供了和 `ggplot2` 比较相似的绘图理念，比 `ggplot2` 更加灵活。首先借用其中的 `grid.text` 为我们的作品添加文字标志，`label` 设定文本内容，`x`、`y` 设定文本的相对位置，注意这里的位置不再是之前所绘图表的坐标轴标定的位置，而是 `grid` 将整张图的长宽通过限度为 1 的纵横网格体系分隔的相对位置，将文字添加在左下角 (0.99, 0.02)，可以尝试修改坐标以了解 `grid` 的坐标体系；然后设置了对齐方式；最后通过 `gpar` 函数设置了标签文字属性，包括大小、字体、颜色等。

输出图表如图 3-34 所示。

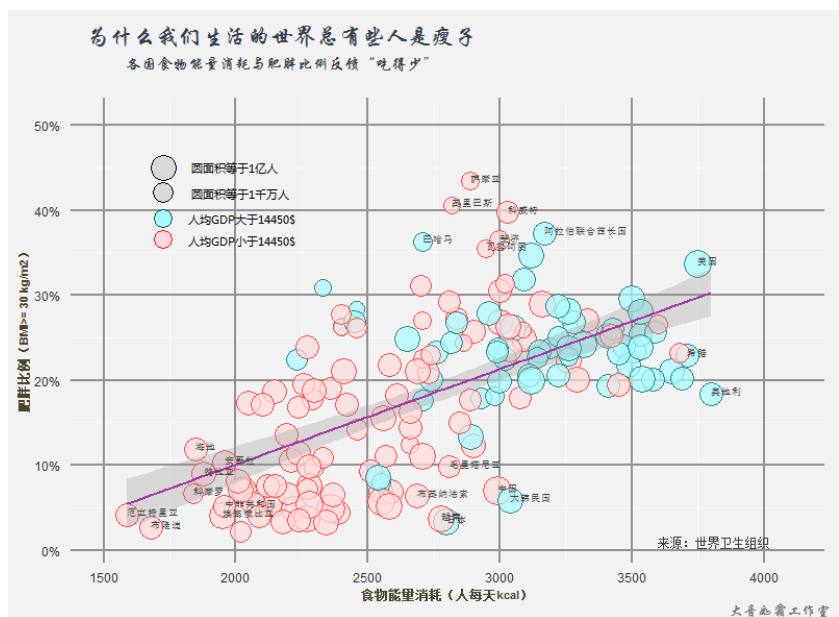


图 3-34

• 保存图片

```
1 png(filename = ("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数据  
可视化说起/plot/ggfinal.png", width = 800, height = 587)  
2 print(p)  
3 add_credits(fontsize = 14)  
4 dev.off()
```

上面的 4 行代码才是图片输出的代码，使用 png 启动 PNG 绘图设备，filename 设置输出地址，width、height 设置长宽。确定长宽时，最好通过缩放图片窗口大小，将图片调整为最佳的大小，然后通过截图工具获得合适的长宽比。

个人风格不是昙花，而是始终统一的个人体系，对于公司也一样。

本节通过一幅图串讲了 ggplot2 的原理和主要的函数、对象体系，同时将之前讲解的图表结构知识用于实践，这里面有很多不足（例如极个别标签重叠的问题，后面章节会讲解处理方式），也许难入大家的法眼，但是生活中有一个有趣的现象：很多事情坚持久了，才能得到大家的谅解、认可和推崇，个人风格也一样，如果是正确的就应该坚持到底。完整代码请见本书下载文件。

3.2.2 基础绘图科学: ggplot2 包的主题函数继承关系图(关系网络图)

其实觉得一门技术难学，是因为还没有入门。新东西之所以为新是因为在某些程度上超越了自己现有的知识体系，难点就在这里了。上节末尾学习了 ggplot2 的主题函数，一时间觉得函数栉比如

鳞，很难记忆。其实还是有两条规律可循的：其一，主题反映的是风格，什么是风格？风格就是独特的、稳定的艺术形式，所以主题函数 `theme` 一旦形成，以后绘图基本上没有太大改动，直接加在代码的末尾即可；其二，主题函数 `theme` 的参数也是有规律可循的，前面已经讲过，等号前面为图表元素名称，等号后面的函数名为图表元素的几何属性，其前后是继承关系，比如 `panel.background` 参数从父函数 `rect` 继承，它的类（class）自然是 `element_rect`，也就是等号后面的函数名称。

不妨绘制一张主题函数的继承关系网络，方便记忆查询，同时初识另一种数据可视化图表：关系网络图，另外，`igraph` 包绘制关系网络图使用 R 的基础绘图函数，也可以学习一下基础绘图函数体系，可谓一石三鸟。`theme` 函数内的参数继承关系均存放在一个叫 `element_tree` 的 list 内，可以通过“`ggplot2:::element_tree`”获取，`:::` 这个函数属于基础函数，用于获取包内的数据。这个 list 的每个元素包括 3 个属性：`class`、`inherit`、`description`，其中 `class` 就是这个参数的类；`inherit` 是它的父函数；做社交关系图只需要函数名和它的父函数这两列就可以了，先把 list 整理一下，如下所示。

- 关系数据整理

```
1 library(plyr)
2 library(igraph)
3 item <- ggplot2:::element_tree
4 edges <- do.call('rbind', item)
5 edges <- transform(edges, child = row.names(edges))
6 edges <- edges[, c("child", "inherit")]
7 edges <- rename(edges, replace = c("inherit" = "parent"))
```

上述代码中，首先将 list 的内容赋值给 `item`。`item` 里所有元素都是一个包含 3 个元素的小 list，所以是等长的，使用 `do.call` 函数将 list 转化为数据框，这个方法仅在 list 的元素长度一样时才起作用。`do.call` 函数包括两个部分：`call` 和 `do`，前者只调用和传参（如下例），不执行，后者执行传参后的函数。理论上 `rbind` 函数可以有  $n$  个参数，即可以合并  $n$  个列数或长度相同的数据框（或 list），`do.call` 就是将 `item` 的所有元素作为参数传递给 `rbind` 并执行。

`lapply` 的结果和 `do.call` 比较相似，但是 `lapply` 使用 list 的每一个元素都要调用一次函数，比较像 `map` 的过程，`do.call` 是把 list 的所有元素作为参数仅调用一次函数，这是它们的本质区别。

如果仅识得使用 `do.call` 函数将 list 转换为 `data.frame`，那真是丢了西瓜捡起芝麻。`do.call` 函数的真正作用是：① 以字符串的方式调用函数；② 以 list 的方式传递参数。这句话看似很难理解，试想一个这样的场景，如果一个来源给了你一个存贮在 `y` 里的函数，你不知道它叫什么，但你知道它的参数都存储在 `x` 内，那是怎么完成传参并执行的过程呢？这才是 `do.call` 函数的真实作用，如下所示。

- `do.call` 的真正用法

```
1 x <- 11.51
2 call("round", x) #传参不执行
3 round(11.51)
4 y <- "round"
5 call(y, x) #黑箱传参
6 x <- as.list(x)
7 do.call(y, x) # 传参执行
```



通过 list 到数据框的转换，获得了一个包含父函数的数据框 edges，它所对应的行名称就是函数名，通过 transform 函数将行名称添加到原数据框中为一新列，并命名为 child，然后从数据框中筛出必需的两列数据，并使用 plyr 包里的 rename 函数将 inherit 重新命名为 parent，这样关系图的两个重要数据就备齐了，如下所示。

- 测试关系网络图

```
1 colnames(edges) <- c("first" , "second")
2 ggnet <- graph.data.frame(edges, directed = TRUE)
3 plot(ggnet, edge.arrow.size = 0.01)
4 V(ggnet)
5 E(ggnet)
```

首先，colnames 函数更改了列名，不更改也可以，graph.data.frame 将数据框的前两列作为关系列表( edge list )或边线列表，这个函数根据关系列表或者节点列表生成网络关系图对象，参数 directed 用来定义关系的方向性，关系是有方向的，比如我关注了你的微博而你没有关注我的微博，那咱们的关系方向即从我指向你；然后通过 V 函数和 E 函数可以查看关系网络中的节点和边线；最后使用基础绘图包的 plot 函数将关系网络可视化，其中使用 edge.arrow.size 参数设置了关系方向箭头的大小。

虽然上述代码生成的图片也是关系网络，先不说美观，中间的 NULL 是什么，所以数据中有特殊值需要处理。模仿别人的数据可视化作品时，无论怎么看重数据格式、结构均不为过。上面看到，用 igraph 包制作关系网络图，数据格式为数据框，结构分为两列，起点列和次点列，一行表示 ggplot2 函数的继承关系 edges (当然也可以添加权重列，社交关系里细讲)，每一个函数表示一个节点 ( vertices )，但这种方法无法正常显示孤立的点，比如某个点和其他任何点都没联系，怎么表示它的关系呢？用 NULL 和 NA 都不行，所以只能使用另外一种方法，即将关系和节点分开，关系 ( edges ) 仍然用原来的数据框格式存储但去除孤立点，而节点全部存储在另外一个单列数据框中，包括孤立点。

- 数据调整

```
1 ver <- data.frame(edges$first)
2 type <- rep(3, length(ver[, 1]))
3 type[edges$first %in% edges$second] <- 2
4 type[which(edges$second == "NULL")] <- 1
5 edges <- edges[-which(edges$second == "NULL"), ]
```

以上代码对数据进行了两个主要调整，首先，第 1 行代码提取了所有的节点作为节点数据框，基本上原数据框的第一类就已经包含了图中将要出现的所有节点，然后对节点分级：一级节点没有父函数；二级节点既可以是子函数又可以作为父函数；三级节点没有子函数。产生一个和节点 ver 等长的等级标记向量 type，每个值均赋为 3，表示三级节点（所有的函数都是子函数）。二级节点就是上面整理的父函数 ( parent )，也就是 second 列，换句话说，所有在 second 列出现的函数均可定义为二级节点，这里囊括了根函数，所以通过 %in% 函数在 second 列中查找第一列的元素，查到了就返回 TRUE，没查到就返回 FALSE ( 结果和第一列长度相同 )，将所有对应行的 type 值改为 2，即二级节点。一级节点就是那些 second 列为 NULL 的函数，which 返回符合表达式的行编号，将对应的 type 赋值为 1。数据清洗整形是很难避免的过程，上面整理数据按照集合从三级到一级逐级缩小的包含关系处理，省了很多麻烦，不信你可以按照集合从小到大筛选调整试试。

为了达到目的，代码延长到了令人心慌的程度，建议停下来在纸上重新画一画思路。

搞定了节点数据框和节点分级，只需要将 edges 数据框中 second 列为 NULL 的行删除就是所需的关系数据框，完成以后就可绘图了。

- 关系网络图

```
1 ggnet <- graph.data.frame(edges, vertices = ver)
2 plot(ggnet, layout = layout.fruchterman.reingold, edge.arrow.size = 0.01,
vertex.size = 4, vertex.label.dist = 0.25)
```

输出效果如图 3-35 所示。

上述代码中，`graph.data.frame` 函数指定了两个参数，`edges` 作为关系数据，参数 `vertices` 由节点数据 `ver` 提供。在绘制关系图时，比上次多了三个参数，`layout` 用于指定关系网络的坐标体系，可以通过?layout 查看帮助文档，后面与 `vertex` 相关的两个参数分别用于指定节点的大小和稍微调节点标签的位置，避免标签和节点重合。

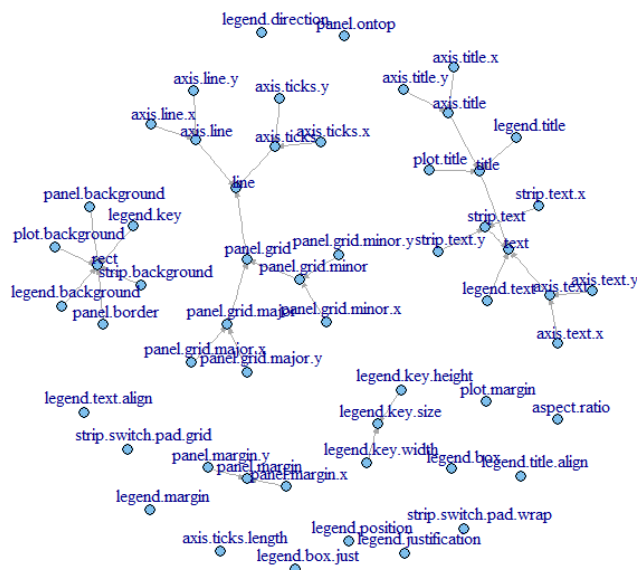


图 3-35

网络图已经完成了，但是很丑，所有引起别人怀疑成果专业性的东西都是我们不愿意看到的，所以这张图还需要加以修饰，例如不同的级别节点需要用不同的颜色加以区分，如下所示。

### ● 修饰关系网络图

```
1 ver <- data.frame(ver, type)
2 names(ver) <- c("first", "type")
3 V(ggnet)$type = as.character(ver$type[match(V(ggnet)$name, ver$first)])
4 V(ggnet)[V(ggnet)$type == "1"]$color <- rgb(112, 48, 160, max = 255, alpha
= 180)
```

```

5 V(ggnet)[V(ggnet)$type == "2"]$color <- rgb(235, 75, 113, max = 255)
6 V(ggnet)[V(ggnet)$type == "3"]$color <- rgb(0, 130, 137, max = 255, alpha
= 180)
7 require("extrafont")
8 loadfonts(device="win")
9 par(mar = c(0,0,2,0), bg = rgb(242, 242, 242, max = 255))
10 plot(ggnet, layout = layout.fruchterman.reingold,
11       edge.arrow.size = 0.01, vertex.color = V(ggnet)$color,
12       vertex.frame.color = NA, vertex.size = 8, vertex.label.dist = 0.35,
13       vertex.label.cex = 0.8, vertex.label.color = V(ggnet)$color)
14 title(main = "ggplot2 主题函数关系网络", cex.main = 1, col.main = rgb(64,
64, 64, max = 255), family = "STXinwei")

```

上述代码中，首先将 `ver` 和相应的 `type` 类型捆绑为数据框，通过 `names` 函数重新命名数据框列名（想一下学习了多少命名函数了）。然后给 `ggnet` 的节点创建一个新属性 `type`，通过 `match` 函数在 `ver$first` 中查找节点名称，返回 `V(ggnet)$name` 在 `ver$first` 中的位置，这一点是 `match` 函数与 `%in%` 函数的不同，根据位置筛选相应的 `ver$type` 并赋值给节点的新属性 `type`，到这里我们已经学了两种筛选匹配函数了，这样通过匹配的方式给每个节点添加了相应的节点级别，也就是 `type` 属性。

根据 `type` 给节点添加新属性 `color`，不同的属性类型赋予不同的颜色值。在 `rgb` 函数中有两种表达三色的方式：其一，红绿蓝三色的范围为 0 至 1，相应的透明度参数的取值范围也为 0 至 1；但是大家更加熟悉的红绿蓝三色范围为 0 至 255，在 `rgb` 函数中也可以设置，不过要添加 `max` 参数并赋值为 255，相应的透明度 `alpha` 参数的取值范围也转变为 0 至 255。

颜色分配完成后，就可以绘制关系网络图了，`igraph` 使用 R 基础绘图函数 `plot` 完成绘图，`plot` 的很多图形参数如图形边界、点线样式、字体、字号、粗细、背景等均可以使用 `par` 函数（Parameters 缩写）进行全局设置，当然也可以在具体的绘图函数（如 `line`、`points` 等）中临时设置，这里使用 `par` 函数设置了图形边界和背景色，如果有读者对 3.2.1 节 `ggplot2` 的图形边界 `plot.margin` 的参数还不理解，不如将上面的图形边界向量的第三个值改为 0（下左上右），向下执行看看发生了什么？没错，边界没有足够的位置显示图表标题。

然后将节点的颜色属性值 `V(ggnet)$color` 指定为节点的颜色 `vertex.color` 参数；节点边框（`vertex.frame.color`）的颜色设置为无色；`vertex.label.cex` 用于指定节点标签的字体大小，`cex`（character expansion）即 R 基础包调节字体大小的参数；为了使标签和节点在视觉上比较统一，将节点颜色属性值指定为节点标签的颜色 `vertex.label.color`。

雁过留声，人过留名，图表的标题不得不加，这里使用 `title` 函数，第 1 个参数为主标题内容；`cex.main` 参数用于指定主标题字体大小，`col.main` 用于指定主标题的颜色；`family` 用于指定主标题的字体，这里使用了 Windows 系统自带的字体华文新魏，所以加载了 `extrafont` 包，如图 3-36 所示。

为什么有些人刚接触 `ggplot2` 就觉得函数太多，其实如果使用 R 的基础绘图包同样会面临函数、参数连篇累牍的局面，之所以没遇到，是因为有些人绘图时太不讲究了。任何一个专业的领域必然有其深不可测的一面，吓退那些投机取巧者，从而给专业主义者留下足够的利益空间，其实一旦深入理解了，以专业的眼光看待它们就变简单了，这就是把书读厚、把书读薄的意思吧。

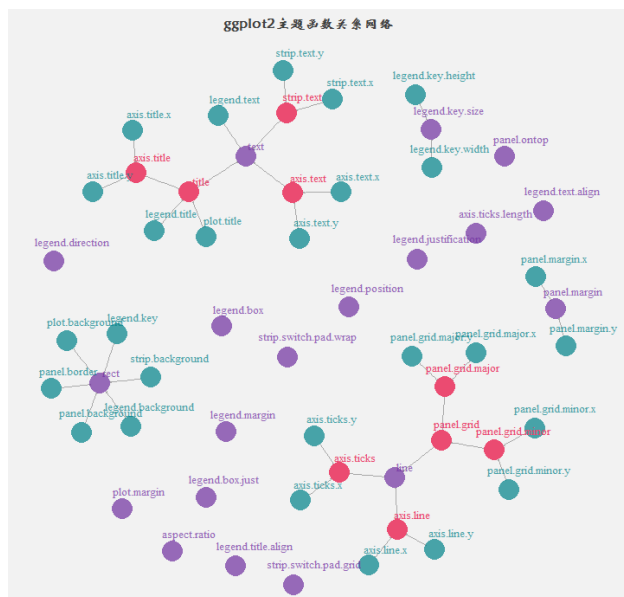


图 3-36

### 3.2.3 基础图表一网打尽

基础图表包括柱形图、条形图、散点图、折线图、极坐标图、面积图、分布图、地图等，特点简洁明了，其他创新图表基本上都是这些图表的变形和组装，比如桑基图（Sankey），我认为就是条形图和面积图的组合。关于数据可视化我们必须达成这样的共识：数据可视化不是将图表和数据复杂化，更不是为了制造炫酷的图表形式，而是为了更加简洁的表达规律、趋势等。因此，图表的组装和综合要从两个方面考虑：

(1) 将规律和指标综合化, 这种综合是内在规律的升级综合, 比如需要使用多个基础图形表达的多个指标, 就要考虑能不能将这些指标通过数据分析、数据挖掘的方式合并为一个综合指标, 也许需要多条折线表达的图形, 一条折线就可以完成了(信息损失基于你的算法), 例如我们既不可能将上证 A 股 1378 只股票的趋势折线绘制在一张图上也都不可能绘制 1378 张折线图, 而是编写了一个上证综合指数反映股票市场的整体走势, 虽然有些股票一时间可能和综指走向相反, 但整体和长期来看都要屈服于上证综合指数的“石榴裙”下;

(2) 通过组装、变形图表改变信息的表达形式, 让分散的基础图表传达的规律一目了然地展现在一张图表中, 比如地图热力图和气泡图结合等。

我倾向于选择第(1)条,只有第(1)条不适宜时才选择第(2)条。

无论选择哪种可视化效果优化的方法，首先还是要学会绘制基础图表，3.2.2 节通过绘制一幅复杂的图表基本上学习了 `ggplot2` 包的各个方面，本节就利用上面的知识，绘制一些走心的基础图表，用以分析历年诺贝尔奖数据。

## 1. 柱形图、条形图、堆积图

继 2012 年莫言获得诺贝尔文学奖后，我国科学家屠呦呦女士又获得了诺贝尔生理学奖，相继打破了土生土长的中国人不可能获得诺贝尔奖的说法，下面不妨看看数据还能揭示哪些对诺贝尔奖的误解。

这里没有使用 R 语言中自带的数据库，原因在于我始终认为使用干净完备的数据实现建模、可视化太容易，无论是数据挖掘还是数据可视化，难点都在于数据清洗、整形和结果解读。所以我们一直坚持使用有意思的 raw data 或者 dirty data，但是这也增加了我们的工作量和本书的篇幅。

- 自编函数、辅助数据

```
1 add_credits = function(fontsize = 12) {
2   grid.text("大音如霜工作室", x = 0.99, y = 0.02, just = "right", gp =
gpar(fontsize = fontsize, col = "#777777", fontfamily = "STXingkai"))
3 }
4 title_with_subtitle = function(title, subtitle = "") {
5   ggtitle(bquote(atop(. (title), atop(. (subtitle))))))
6 }
7 library(grid)
8 library(extrafont)
# font_import()#引入字体
9 loadfonts(device = "win")
10 cate <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数
据可视化说起/data/category.csv", header = T, sep = ",", stringsAsFactors =FALSE)
11 cncountry <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气
质的数据可视化说起//data/ISO_3166_1.csv", header = T, sep = ",", stringsAsFactors
=FALSE)
```

上述代码中载入添加公司标志和主副标题的自编函数及系统字体，同时载入两个辅助数据集，其中 category 数据集为了将英文的诺贝尔奖类别替换为中文，ISO\_3166\_1 数据集很有用，它包含了世界各国的英文双字符、三字符简称、英文名称、官方全称和中文名称，其作用就是将各国英文称呼替换为中文。

- 载入数据

```
1 library(jsonlite)
2 url <- "http://api.nobelprize.org/v1/laureate.json"
3 nobel <- fromJSON(url)
4 class(nobel)
5 names(nobel)
6 names(nobel$laureates)
7 names(nobel$laureates$prizes[[1]])
```

上述代码中，通过诺奖官网 API 获得的获奖者 (laureates) 数据为 JSON 格式，因此需要 jsonlite 包中的 fromJSON 函数将 JSON 格式转化为 R 对象 list 类型 (通过 class 查看对象类型)。转化之后该 list 包含一个名为 laureates (获奖者) 的元素，元素 laureates 是一个包含 13 列的数据框，每行指向一名获奖人，包括获奖人的姓名、出生年月、国家等属性信息；数据框的最后一列 prizes，是获奖人获得奖章的属性信息，有些人可能多次获奖，所以 prizes 列又是一个 list，它的每一个元素却是一个

5 列的数据框，包括每一个奖章，诸如获奖时间、奖章类别等信息，现在你应该对 R 的数据结构有更深刻的认识了，这个数据就是 list 里面嵌套数据框，数据框里又嵌套 list，然后 list 里再嵌套数据框的结构。

先看看获奖者出生地的分布，换句话说就是各国的获奖次数，这里是人次，不是人数，人数按原数据框 laureates 的行分组统计就行，但是次数就比较麻烦，比如一个人获奖两次，那就要统计两次，可是数据框 laureates 每行代表一个获奖者，其国家属性也只有一次，需要将国家列按照 prizes 的元素的长短复制延长，如下所示。

### • 数据调整 (1)

```
1 library(plyr)
2 prizes <- ldply(nobel$laureates$prizes, as.data.frame)
3 cnt <- sapply(nobel$laureates$prizes, function(x) nrow(x))
4 nobel$laureates[cnt > 1, ]
5 prizes$country <- rep(nobel$laureates$bornCountryCode, cnt)
6 prizes$id <- rep(nobel$laureates$id, cnt)
7 prizes$gender <- rep(nobel$laureates$gender, cnt)
8 p3 <- as.data.frame(table(prizes$category, prizes$country), stringsAsFactors
= FALSE)
```

plyr 包里有高效整理数据的 apply 家族函数，ldply 函数即是 list、data.frame、apply 的缩写，上述代码使用 plyr 包里的 ldply 函数将 list 转化为数据框 prizes，前面的章节我们使用 do.call 函数完成了相同的动作，因为一个人可以多次获奖，数据框 prizes 比数据框 laureates 长，需要将 laureates 某些列，比如出生地扩展成和 prizes 等长，并一一对应。使用匿名函数 function(x) nrow(x) 统计 laureates\$prizes 里每个小数据框的行数，所谓匿名函数就是没有名称的临时函数，sapply 函数将 list 的每一个元素应用于后面的匿名函数。cnt 存储了每一个获奖者的获奖次数，可以通过 cnt 大于 1 筛选 laureates 中哪些人多次获奖。因为 cnt 和 laureates 是等长的，laureates 中每一个获奖者的 bornCountryCode 也是和 cnt 等长一一对应的，可以使用 rep 函数将每一个 bornCountryCode 按照 cnt 对应的次数复制，然后赋值给数据框 prizes 的新列 country 即可（这一过程直接完成了创建新列和赋值，可与 transform 函数比较），这样每一枚奖章均有了对应的获奖人出生地。使用同样的方法给每一枚奖章添加获奖人的 id 编号和性别，大家一定要熟练这个扩展数据框的过程。然后使用 table 函数统计 prizes 数据框中不同国家各个类别出现的次数即可。

### • 数据调整 (2)

```
1 library(dplyr)
2 temp <- aggregate(Freq ~ Var2, data = p3, sum) %>% rename(total = Freq)
3 p3 <- p3 %>% left_join(cncountry[, c("Alpha_2", "simpchinese")], by =
c("Var2" = "Alpha_2")) %>% left_join(temp) %>% left_join(cate)
4 p3 <- p3[order(p3$total, decreasing = TRUE),]
5 p3$simpchinese <- factor(p3$simpchinese, levels = unique(p3$simpchinese))
```

熟悉 Excel 透视表的读者就很容易理解 aggregate 函数，其作用就是制作透视表。上述代码中，第 2 行代码使用 aggregate 函数，第 1 个参数为方程参数，Freq ~ Var2 使用 Var2 对 Freq 分组，第 2 个参数指定数据集，第 3 个参数指定函数名，这里使用 sum 函数对分好组的 Freq 求和；%>% 来自 dplyr 包的管道函数，其作用是将前一步的结果直接传参给下一步的函数，从而省略了中间的赋值步

骤,可以大量减少内存中的对象,节省内存,可惜的是应用范围还不是很广;其后 `rename` 更改列名。

第3行代码需要将原数据中的国家英文缩写匹配为中文, `left_join` 同样来自 `dplyr` 包,其作用是完成左关联,筛选出备用数据框 `cncountry` 的 `Alpha_2` 和 `simpchinese` 两列与 `p3` 关联,参数 `by` 指定关联的匹配列,即 `p3` 的 `Var2` 列和 `cncountry` 的 `Alpha_2` 匹配;后面将刚刚计算的 `temp` 数据框关联上去,这里没有指定匹配列, `left_join` 函数将使用两个数据框中同名列进行关联匹配,最后给 `p3` 匹配上了奖章的中文分类。

绘图时,为了方便解读和美观,一般会将数据按照一定的规律排列,比如我们希望国家按获奖次数由高到低排列,所以需要用 `order` 函数降序排列(`decreasing = TRUE`)(见第4行代码),为什么降序,因为制作的条形图、柱形图, `X` 轴一般为分类变量, `ggplot2` 比较特别,其他绘图系统可能也存在这种情况,它不会根据分类变量对应的数值排序,而是根据分类变量的因子水平先后顺序进行排序的,因子水平靠前者将被排在前面。所以排序后要使用 `factor` 函数重新定义 `simpchinese` 变量因子水平,到这里数据调整已经基本完成可以尝试作图了。

- 载入公用主题

```
1 library(ggplot2)
2 theme1 <- theme(panel.background = element_rect(fill = rgb(red = 242, green =
242, blue = 242, max = 255)),
3               plot.background = element_rect(fill = rgb(red = 242, green
= 242, blue = 242, max = 255)),
4               plot.title = element_text(size = rel(1.2), family = "STXingkai",
face = "bold", hjust = 0.5, colour = "#3B3B3B"),
5               panel.grid.major = element_line(colour=rgb(red = 146, green
= 146, blue = 146, max = 255),size=.75),
6               panel.border = element_rect(colour = rgb(red = 242, green = 242,
blue = 242, max = 255)),
7               axis.ticks = element_blank(),
8               axis.text.x = element_text(colour = "grey20", size = 8),
9               axis.text.y = element_text(colour = "grey20", size = 10),
10              axis.title.y = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = 0.5),
11              axis.title.x = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = -0.5),
12              legend.background = element_rect(fill = rgb(red = 242,green
= 242, blue = 242, max = 255)),
13              legend.position = "bottom")
```

主题一般很少改变,所以我们首先设置一个公用的主题备用(见上述代码),主题里的知识已经讲过多次了,在这里不再说明。

条形图分为4种形式:柱形图、条形图、堆积图、簇状图,堆积图又分为数值堆积图和百分比堆积图,下面依据数据简单实现一遍。

- 柱形图

```
1 temp <- unique(p3[, c("simpchinese", "total")])
```

```

2 temp <- temp[1:20,]
3 p <- ggplot(temp, aes(simpchinese, total, label = total)) +
4   geom_bar(stat = "identity", fill = rgb(0, 130, 137, max = 255, alpha =
200)) +
5   geom_text(vjust = 1, color = "white") +
6   ylab("获奖人次") +
7   xlab("") +
8   title_with_subtitle("诺贝尔奖获得者出生地统计") +
9   theme_bw(18) + theme1 +
10  theme(axis.text.x = element_text(size = 9, angle=90, hjust = 1, vjust
= 0))

```

首先提取国家名称、获奖次数去重后作为临时数据框 temp（仅绘制前 20 名国家），ggplot 指定数据集 temp、横坐标 simpchinese、纵坐标 total、数据标签 total 等映射，geom\_bar 添加几何形状，ylab、xlab 设置纵横坐标名称；然后添加主副标题，前面均已经详细介绍，不再赘述；添加固定的主题，对主题需要简单修改，因为 X 轴刻度标签过长，横向摆放重叠，所以单独对主题里的 X 轴标签进行修改，仍然使用 theme 函数，将标签的角度调整为 90 度即可，然后将标签调为右侧对齐和下侧对齐。效果如图 3-37 所示。

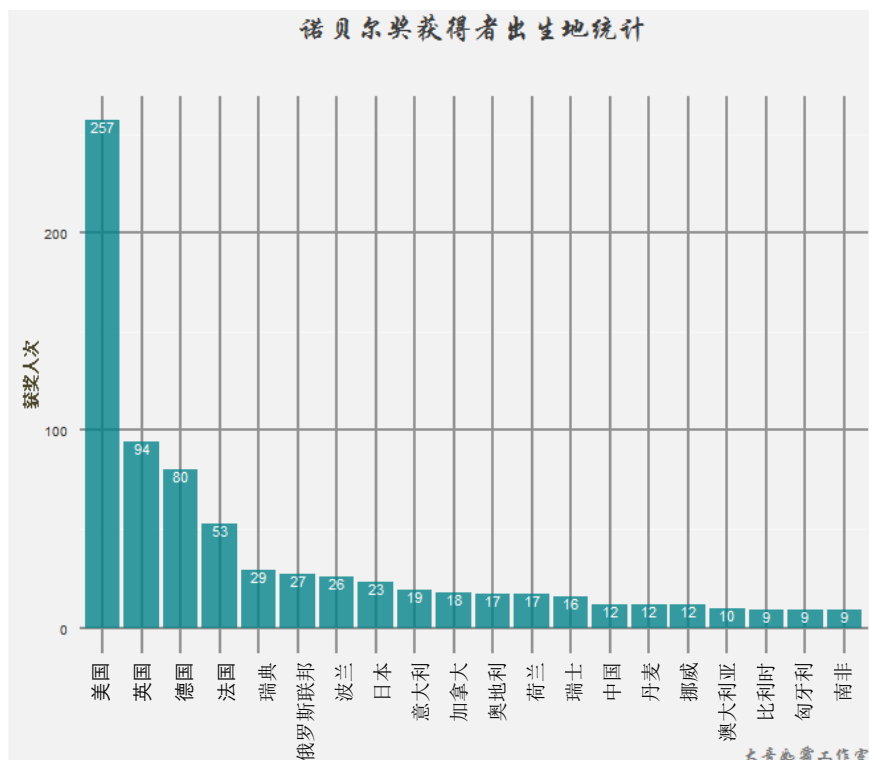


图 3-37



需要注意的是，我们使用了映射的方式为数据添加了标签，首先将数据值映射为 label，然后通过 geom\_text 添加标签和设置标签的属性，但是不建议通过全局的方式添加标签，原因很简单，没必要将所有数据点的标签都标记出来，那还要刻度线和坐标轴何用？标签是为了突出重点的数据点而存在的。

• 条形图

```
1 p <- ggplot(temp, aes(simpchinese, total)) +  
2   geom_bar(stat = "identity", fill = rgb(0, 130, 137, max = 255, alpha =  
3     200)) +  
4   coord_flip() +  
5   ylab("") +  
6   xlab("") +  
7   title_with_subtitle("诺贝尔奖获得者出生地统计") +  
8   theme_bw(18) + theme1
```

条形图就是柱形图旋转一下就可以了，所以只需添加一个 coord\_flip 函数，flip 为翻转的意思，即将映射旋转一下，但是图表变成了降序排列，可以将原数据的因子水平按照升序排列，再作图就变为国家获奖次数的降序图表了，如图 3-38 所示。

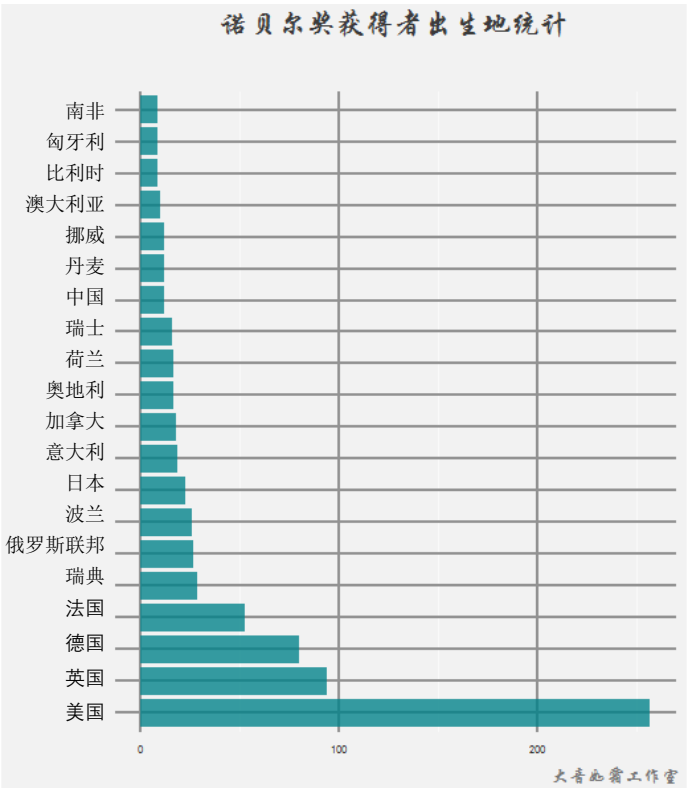


图 3-38

那每个国家获得的奖章类别有什么区别呢？这个问题可以使用堆积图和簇状图解决。

- 堆积图

```
1 p3 <- p3[order(p3$total),]
2 p3$simpchinese <- factor(p3$simpchinese, levels = unique(p3$simpchinese))
3 p <- ggplot(p3, aes(simpchinese, Freq, fill = cat)) + geom_bar(stat =
"identity", position = "stack") +
4 coord_flip() +
5 scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
"#2c7fb8", "#253494"), name = "图例") +
6 ylab("") +
7 xlab("") +
8 title_with_subtitle("诺贝尔奖获得者出生地统计") +
9 theme_bw(18) + theme1
```

效果如图 3-39 所示。

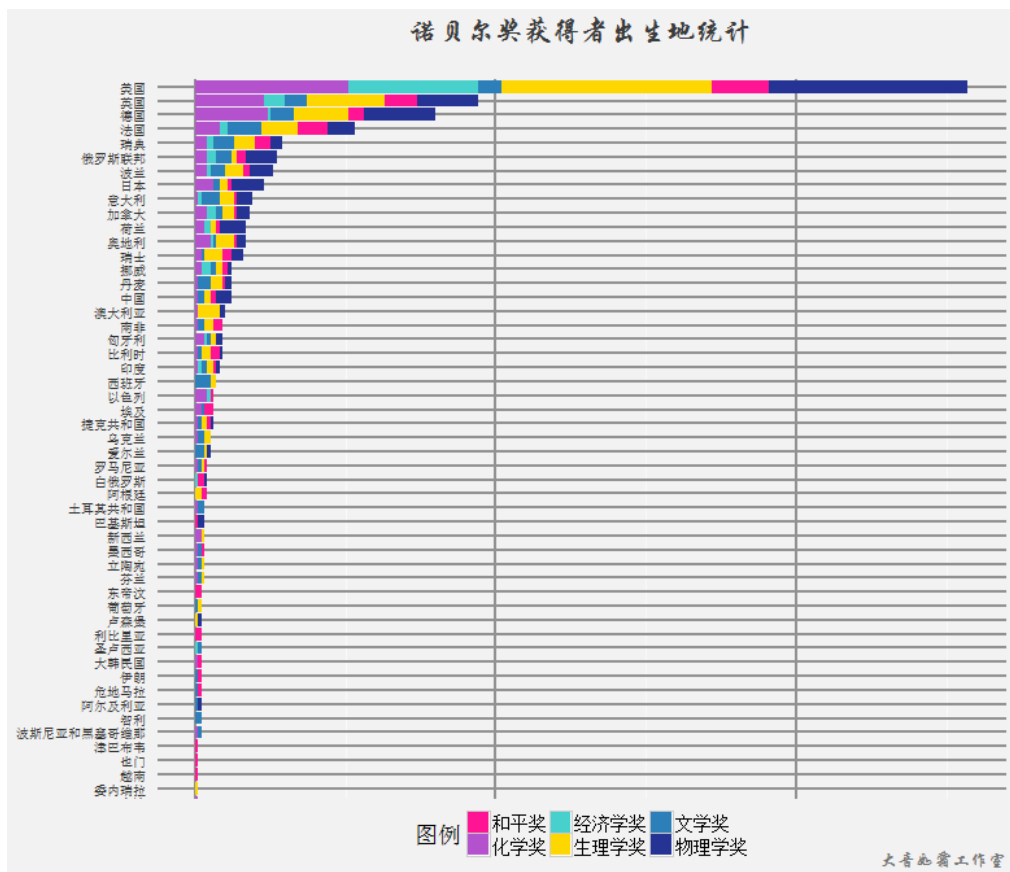


图 3-39

为了图表中的条形按照降序排序，对原来的数据框 p3 重新按照升序进行排序，然后重新设定因子水平顺序；ggplot 函数指定了新的数据集和映射关系，将 Freq 映射为 Y 轴，同时使用 cat 列进行填充或者称为分组，然后 geom\_bar 添加几何形状图层，只不过这里添加了一个参数，用于指定条形的堆积形式：stack 表示上下堆积，dogge 表示簇状堆积，fill 表示百分比堆积。其他只是使用 scale\_fill\_manual 离散标尺函数为 6 种诺奖设置了不同的颜色，前面也已经讲过了。

• 簇状图

```
1 temp <- p3[length(p3[, 1]):415,]
2 p <- ggplot(temp, aes(simpchinese, Freq, fill = cat)) + geom_bar(stat =
"identity", position = "dodge") +
  #coord_flip() +
3 scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
  "#2c7fb8", "#253494"), name = "图例") +
4 ylab("获奖人次") +
5 xlab("") +
6 title_with_subtitle("诺贝尔奖获得者出生地统计") +
7 theme_bw(18) + theme1
```

效果如图 3-40 所示。

由于簇状图会大大延长横坐标，所以选取了获奖前 7 名的国家的数据，length 函数计算 p3 的第一列有多长，然后从此选取到 415 行就是前 7 名的国家；作图的内容几乎没有改变，只是将几何形状 geom\_bar 函数的位置参数改为 “dodge”，同时注销了坐标轴转换的函数。与条形堆积图比较，我觉得簇状图是一种即将消失的图表类型，因为占据空间大，对比也不方便。

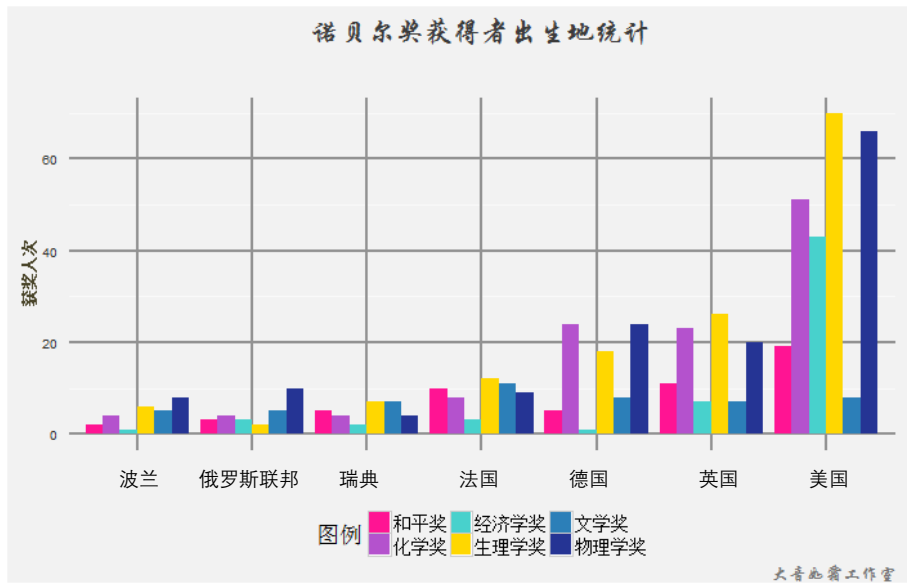


图 3-40

- 百分比堆积图

```
1 library(scales)
2 p <- ggplot(p3, aes(simpchinese, Freq, fill = cat)) + geom_bar(stat =
"identity", position = "fill") +
3   coord_flip() +
4   scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
"#2c7fb8", "#253494"), name = "图例") +
5   scale_y_continuous(labels = percent) +
6   ylab("") +
7   xlab("") +
8   title_with_subtitle("诺贝尔奖获得者出生地统计") +
9   theme_bw(18) + theme1
```

上述代码只是将几何形状 `geom_bar` 函数的位置参数改为了“fill”，就基本完成了任务，`scale_y_continuous` 连续变量标尺函数需要使用 `scales` 包的 `percent` 函数将 *Y* 轴标签改为百分比形式。其实如果不知道这种做百分比堆积图的方法也可以通过调整数据来实现，将各国各类奖项除以各国获奖次数的总和，使用所得数据作条形堆积图即可，所以凡事不必拘泥于一点。

## 2. 折线图、面积堆积图、散点图、气泡图

折线图是观察趋势的不二法门，历年来诺贝尔奖获奖人的性别有什么趋势呢？首先 `table` 函数通过对行计数统计每年不同性别的获奖人次，你也可以使用 `aggregate` 函数制作透视表来完成；然后将 `p3$Var1` 数据类型更改为数值型，数据就准备好了，如下所示。

- 折线图

```
1 p4 <- as.data.frame(table(prizes$year, prizes$gender), stringsAsFactors
= FALSE)
2 p4$Var1 <- as.numeric(p4$Var1)
3 p <- ggplot(subset(p4, Var2 != "org"), aes(x = Var1, y = Freq, color = Var2)) +
4   geom_line(size=1.3) +
5   annotate("text", x = 2006, y = 13, colour = "#008B8B", label = "男士",
size = 4, fontface = "bold") +
6   annotate("text", x = 2006, y = 6, colour = "#FF1493", label = "女士", size
= 4, fontface = "bold") +
7   scale_color_manual(values = c("#FF1493", "#008B8B")) +
8   scale_x_continuous(breaks = seq(1900, 2015, by = 10)) +
9   ylab("人次") +
10  xlab("年份") +
11  title_with_subtitle("诺贝尔奖获得者性别趋势") +
12  theme_bw(18) + theme1 +
13  theme(legend.position = "")
```

上述代码中，首先使用 `ggplot` 函数中的 `subset` 函数将获奖者性别为组织（org）的行剔除，仅剩余个人获奖者；同时 `aes` 指定了坐标轴、颜色分组映射，这里已经使用 `Var2` 同时执行了分组绘制和颜色分组，不用多余地指定 `group` 参数；然后使用 `geom_line` 添加折线图层，由于只有两条曲线，没

必要使用默认图例解释不同颜色代表什么性别，不如使用 `annotate` 直接在曲线上标注；借用 `scale_x_continuous` 函数的 `break` 参数设定 X 轴的刻度区间，避免刻度标签重叠；最后既然要移除默认图例，就需要对原来的主题设置稍作修改，将 `legend.position` 指定为空。效果如图 3-41 所示。

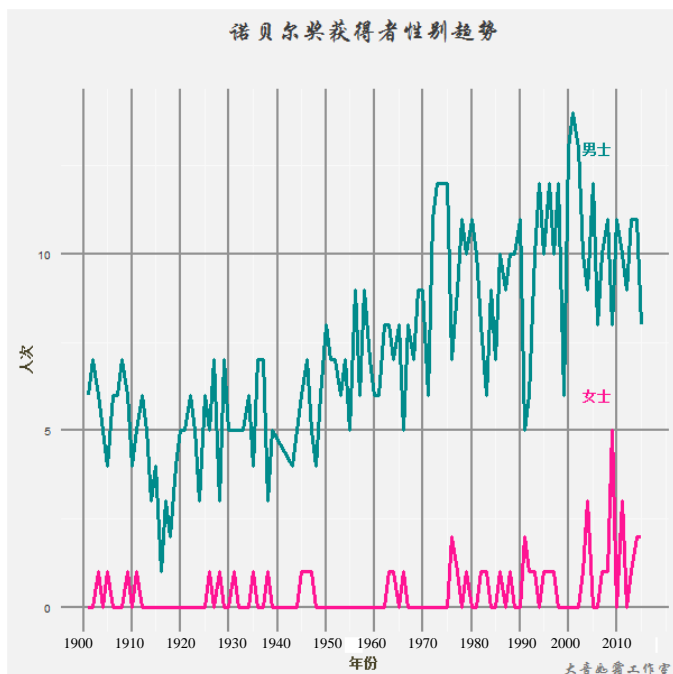


图 3-41

其实就诺贝尔奖经常缺少女士得主的情况，我认为使用折线图不如使用面积堆积图更加合适，如下所示。

- 面积堆积图

```
1 p <- ggplot(subset(p4, Var2 != "org"), aes(x = Var1, y = Freq, fill= Var2)) +
2   geom_area(position = 'stack') +
3   annotate("text", x = 2006, y = 13, colour = "#008B8B", label = "男士",
4     size = 4, fontface = "bold") +
5   annotate("text", x = 2006, y = 6, colour = "#FF1493", label = "女士", size
6     = 4, fontface = "bold") +
7   scale_fill_manual(values = c("#FF1493", "#008B8B")) +
8   scale_x_continuous(breaks = seq(1900, 2015, by = 10)) +
9   ylab("人次") +
10  xlab("年份") +
11  title_with_subtitle("诺贝尔奖获得者性别趋势") +
12  theme_bw(18) + theme1 +
13  theme(legend.position = "")
```

上述代码中,数据、函数几乎没有修改,除了将 `geom_line` 函数替换为 `geom_area` 图层函数以外,只是将颜色 (`color`) 映射改为填充 (`fill`) 映射,将颜色标尺改为填充标尺 `scale_fill_manual`。效果如图 3-42 所示。

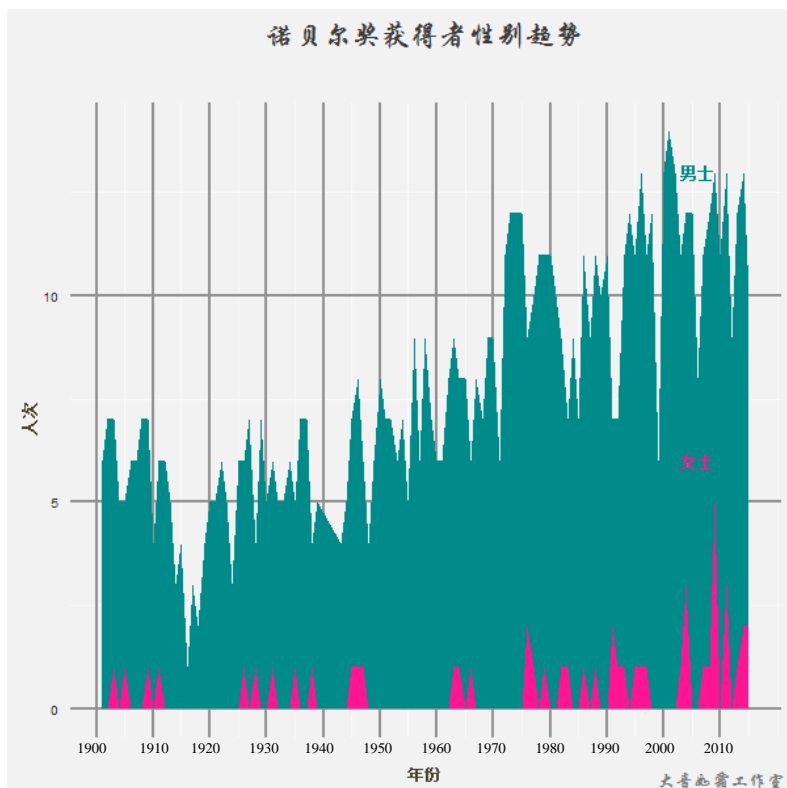


图 3-42

散点图其实前面已经花了大量篇幅来介绍,为了探究诺贝尔奖得主年龄的问题,看看诺奖得主获奖时的年龄趋势,这里再实现一次。

- 数据调整 (3)

```
1 prizes$born <- rep(nobel$laureates$born, cnt)
2 prizes$age <- as.Date(paste(prizes$year, "12-31", sep = "-"), format =
"%Y-%m-%d") - as.Date(prizes$born, format = "%Y-%m-%d")
3 p5 <- left_join(prizes, cate, by = c("category" = "Var1"))
```

上述代码中,首先要将诺奖得主出生年龄对应拓展为和 `prizes` 一样长的向量,然后计算获奖者获奖时的年龄,因为获奖年份只有四位,不到具体的日期,所以使用 `paste` 函数统一添加为 12 月 31 日,并使用 “-” 隔开 (注意 `sep` 和 `collapse` 的区别),使用 `as.Date` 函数将变量类型由字符更

改为日期类型,as.Date 第2个参数用于指定日期格式,最后使用左连接将奖章的中文类别关联上去。

- 散点图

```
1 p <- ggplot(p5[!is.na(p5$category), ]) + geom_point(aes(year, as.numeric(
  age)/365, color = cat)) +
2   stat_smooth(aes(year, as.numeric(age)/365, group = 1)) +
3   facet_wrap(~cat) +
4   scale_x_discrete(breaks = seq(1900, 2015, 25)) +
5   scale_color_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
  "#2C7FB8", "#253494")) +
6   ylab("年龄") +
7   xlab("年份") +
8   title_with_subtitle("诺贝尔奖获得者不同领域的年龄趋势") +
9   theme_bw(18) + theme1 +
10  theme(legend.position = "")
```

上述代码中,首先筛选分类非空的数据,请注意,这里 ggplot 函数没有指定全局映射,而是将映射放到相应的图层中指定,geom\_point 函数里通过 aes 指定了坐标轴和颜色分组变量,并按一年365天将 age 天数换算为年;然后使用 stat\_smooth 函数添加了拟合曲线,它的其他参数已经介绍过了,当你已经对数据分组后,又想以整体拟合,而不是分组拟合,那么需要将 group 参数设置为1。

到这里我们将接触 ggplot2 的最后一个重要概念:facet 分面,为什么分面?因为 ggplot 不允许一图双坐标轴,分面有两种:网格型(facet\_grid)和封面型(facet\_wrap),简单理解,前者是一个二维的网格,各个小分区图的位置是按照指定的二维坐标分布的,后者则生成面板后封装成一个假二维分布图(见下面代码示例),其区别主要在于前者小分区图的行列数根据分组变量已经确定,不能改动,而后者可以通过 ncol 和 nrow 参数设置行列数,这里使用 facet\_wrap 函数根据 cat 变量对图表分面封装。

因为 year 变量为字符型,所以使用 scale\_x\_discrete 标尺设定了 X 轴刻度区间,回忆一下,在绘制折线图时,使用更改变量类型的方法设定了刻度区间,殊途同归而已。其余代码基本没有变动,均是熟悉的内容。

- 分面示例

```
1 p <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
#facet_grid与facet_wrap的区别
2 p + facet_grid(vs ~ am)
3 p + facet_wrap(vs ~ am, ncol = 2)
```

效果如图 3-43 所示。

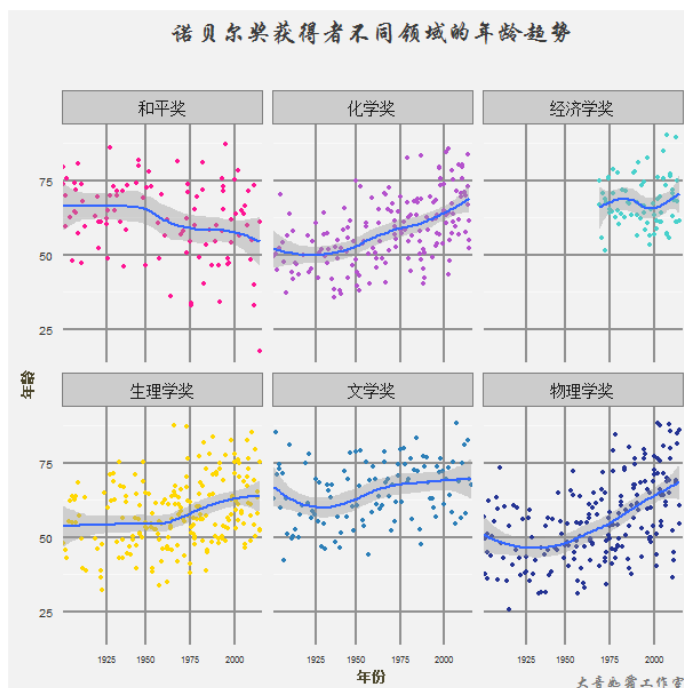


图 3-43

### 3. 气泡图

前面已经花费大量的篇幅讲解，这里不再赘述。

### 4. 分布图（箱线图及其变形）

我们了解数据的目的自然是为了对某种现象的解释和预测，而探索数据就是帮助我们了解数据，我认为了解数据的分布除了解释作用以外，更重要的是在数据中寻找位置，比如想知道有百分之几的获奖得主年龄小于莫言或者屠呦呦？想知道股票低于某一价格的概率有多大？这其实就是在数据中寻找一个合理而恰当的位置。

密度曲线直方图应该是最棒的审读数据分布的可视化方式了，结合了密度曲线和直方图的优点，首先要将 age 列由日期型转化为数值型，同时由单位为天换算为年。

#### • 密度曲线直方图

```
1 p5$age <- as.numeric(p5$age)/365
2 p <- ggplot(p5[!is.na(p5$category), ], aes(x = age)) +
3   geom_histogram(aes(y=..density..), binwidth = 4, colour = "red", fill
= "white") +
4   geom_density(alpha=.4, fill = rgb(red = 0, green = 130, blue = 137, max
= 255)) +
5   geom_vline(aes(xintercept = 57), color="red", linetype="dashed", size=1) +
```



```

6  annotate("text", x = 58, y = 0.01, colour = "#008B8B", label = "莫言的
57岁", size = 4, fontface = "bold", hjust = 0) +
7  title_with_subtitle("诺奖得主年龄分布", "莫言在分布中只能算中年") +
8  ylab("概率密度") +
9  xlab("年龄") +
10 theme_bw(18) + theme1

```

上述代码中，首先筛选分类非空的数据，将 age 映射到 X 轴，geom\_histogram 前面已经说过，用于绘制直方图，直方图分为包括频次和频率，只不过这里加了一个映射 “..density..”，ggplot2 中双 dots 表示对某对象进行统计转换的意思，这里表示对原来的 age 求频率，然后映射到 Y 轴，下面的 binwidth（步长）之类的参数就早已讲过了，需要说明的是为什么做统计变换，因为 geom\_histogram 默认计算所有分段的频次，范围大于或等于零，而密度曲线的范围为 0~1，所以二者的标尺不一样，不可能在同一个 Y 轴上展示，所以将默认的频次通过 density 函数转换为范围为 0~1 的频率，你可以尝试将 aes(y=..density..)这句去掉观察 Y 轴的刻度标尺变化，以加深理解。

使用 geom\_density 添加密度曲线图层，设置了透明度和颜色；geom\_vline 添加一条垂直于 X 轴的直线图层用来标记莫言获得诺贝尔奖时的年龄，其中映射 xintercept 用于设置 X 轴的截距。后面的函数读者很熟悉了。效果如图 3-44 所示。

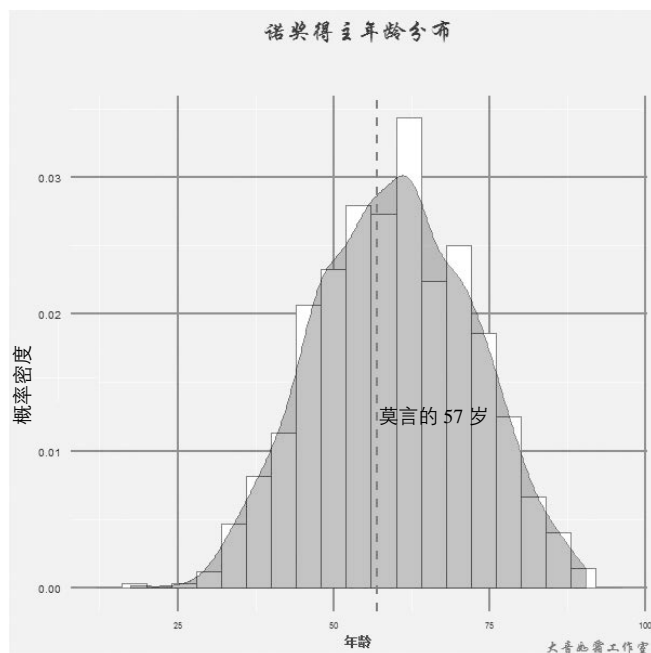


图 3-44

另外一种审查数据分布的图表即箱线图，对于统计学知识扎实的读者来说不失为一种好办法，但是对于普通大众来说，里面涉及的统计概念太多，并不建议在 BI 系统中使用，与其使用箱线图，

不如直接告诉受众百分之几的人年龄低于某条线。

一个善于讲故事的人绝对不会强迫受众理解自己的叙述语言，而是尽量以受众的语言讲述故事。

箱线图代码如下。

- 箱线图

```
1 p <- ggplot(p5[!is.na(p5$category), ], aes(x = cat, y = age, fill = cat)) +  
2   geom_boxplot() +  
3   coord_flip() +  
4   scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",  
5 "#2c7fb8", "#253494")) +  
6   ylab("年龄") +  
7   xlab("") +  
8   title_with_subtitle("诺贝尔奖获得者不同领域的年龄情况") +  
9   theme_bw(18) + theme1 +  
10  theme(legend.position = "")
```

上面的代码基本上都是熟悉的内容，只是添加了新的图层函数 `geom_boxplot`，用于绘制箱线图。效果如图 3-45 所示。

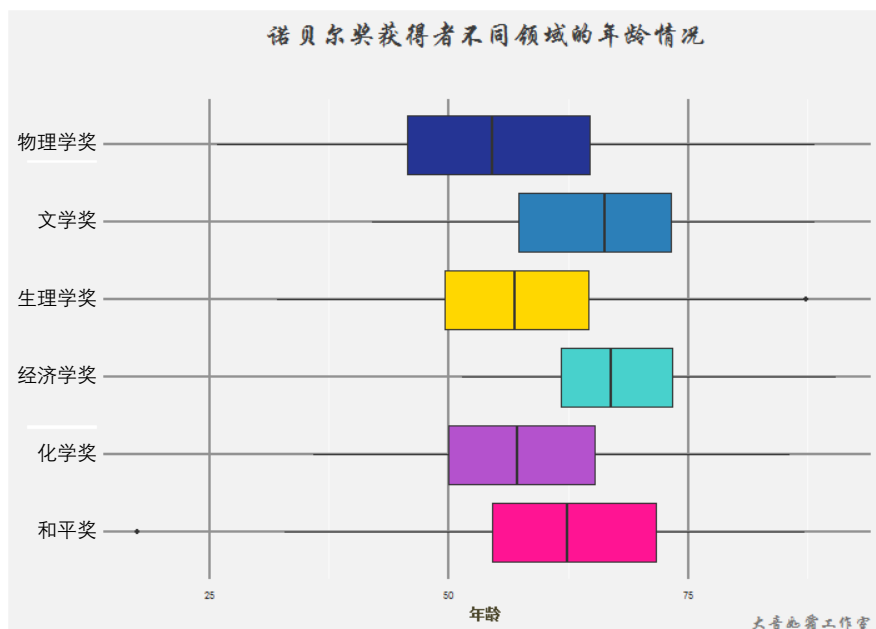


图 3-45

- 箱线图变形

```
1 p <- ggplot(p5[!is.na(p5$category), ]) + geom_violin(aes(cat, age, fill
```

```

= cat)) + stat_summary(aes(cat, age), fun.y = "median", geom = "point") +
  2   scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
"#2c7fb8", "#253494")) +
  3   ylab("年龄") +
  4   xlab("") +
  5   title_with_subtitle("诺贝尔奖获得者不同领域的年龄情况") +
  6   theme_bw(18) + theme1 +
  7   theme(legend.position = "")

```

上述代码中，首先使用 `geom_violin` 绘制小提琴箱线图图层，所指定的映射和箱线图一样；然后使用了 `stat_summary` 函数，指定映射 `cat` 为 *X* 轴，`age` 为 *Y* 轴，并对 *Y* 轴 `age` 求中位数；最后 `geom` 添加中位数点图层；后面的代码基本上没有改变。小提琴箱线图与箱线图的不同在于其边线起伏代表数据的分布，峰代表频率高，谷表示频率低。现在知道为什么我说 `ggplot2` 真正的优势在于其统计变换了吧！效果如图 3-46 所示。

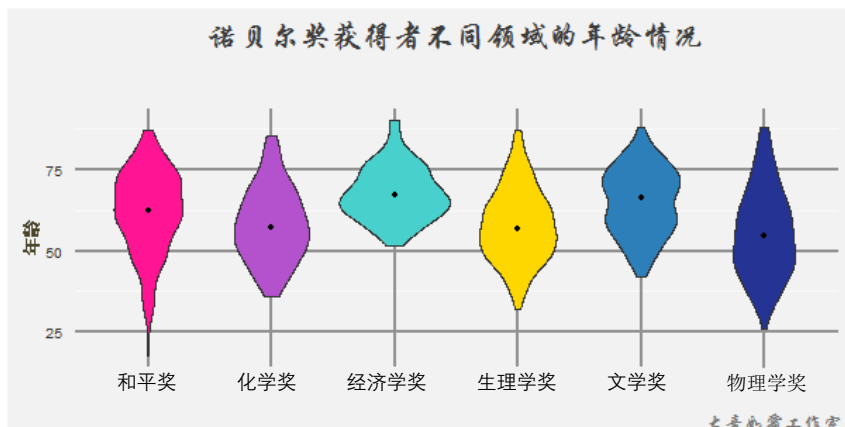


图 3-46

### 5. 极坐标图（饼图、玫瑰图、环图、雷达图）

从平面内的一点出发，围绕该点形成圆，使用这些圆的弧度、半径、面积进行数据可视化统称为极坐标图，比较常见的有饼图、玫瑰图、雷达图。至于这种图的优劣势咱们暂且不提，毕竟还是广大人民群众喜闻乐见的图表，先实现再说。

#### 饼图

看看不同学科获奖人次占总人次的比例，关于比例的数据使用饼图应该会比较合适的选择（有些情况并非如此）。由于极坐标图的坐标体系和其他图类型迥异，需要将太多的部件设置为空，所以不得不准备极坐标图专用的主题（`theme`），然后使用 `table` 计算各个类别的获奖人次，如下所示。

#### • 饼图主题

```

1 pietheme <- theme(plot.margin = unit(c(1, 1, 1.25, 0.5), "lines"),
2   panel.border = element_blank(),

```

```

3     panel.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
4     plot.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
5     plot.title = element_text(size = rel(1.2), family = 'STXingkai' ,
hjust = 0.5, vjust = 0, face = 'bold', colour = '#3B3B3B'),
6     panel.grid = element_blank(),
7     axis.ticks = element_blank(),
8     axis.text.x = element_blank(),
9     axis.text.y = element_blank(),
10    axis.title.y = element_blank(),
11    axis.title.x = element_blank(),
12    legend.position = "bottom",
13    legend.title = element_blank(),
14    legend.background = element_rect(fill = rgb(red = 242, green = 242,
blue = 242, max = 255), size = .1),
15    legend.text = element_text(size = 10, face = "bold"))
16 p6 <- as.data.frame(table(p5$cat), stringsAsFactors = FALSE)

```

在函数 `ggplot` 里，只有一个分类维度下饼图  $X$  轴映射设为空，数值变量被映射给了  $Y$  轴，分类变量映射为填充值 (`fill`)。添加第一个图层是 `geom_bar`，为什么？可以参见前面的百分比堆积图，其实饼图就是百分比堆积图沿  $Y$  轴自旋而成的，这里多了一个 `width` 参数，它在 `bar` 图中用于设置条形的宽度，取值范围为 0~1，转换之后用于设定 `pie` 的半径，视觉上就是饼图到环图的转换；使用 `coord_polar` 函数旋转  $Y$  轴，`start` 参数用于设置 `pie` 图的起始角度，比如 2 表示从 2 点方向起始；其余的代码都比较熟悉，仅 `geom_text` 函数添加标签进行了一层计算，`c(0,cumsum(Freq)[-length(Freq)])` 生成一个和 `Freq` 等长的向量，`cumsum` 用于累加求和，基本上是各类的分界线，但是我们希望标签能标在类别的中线上，所以在后面又加了各类别的 1/2，之后标签仍然使用 `percent` 转换为百分比。这里我们添加标签的方式是直接在 `geom_text` 函数指定标签映射，而不是在 `ggplot` 函数里通过 `label` 参数指定。

### • 饼图

```

1 p <- ggplot(p6, aes(x = "", y = Freq, fill = Var1)) +
2   geom_bar(width = 1, stat = "identity") +
3   coord_polar("y", start = 0) +
4   scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#FFD700",
"#2c7fb8", "#253494")) +
5   geom_text(aes(y = c(0,cumsum(Freq)[-length(Freq)]) + Freq/2,
6     label = percent(Freq/sum(p6$Freq))), size = 9, colour =
"white", vjust = 0.5) +
7   title_with_subtitle("诺奖各类别获奖人次占比") +
8   theme_bw(18) + pietheme

```

### 环图

仅仅需要调节饼图的 `width` 参数，比如 0.2，另外将标签的字体大小与之适应即可。

饼图和环图的效果如图 3-47 所示。

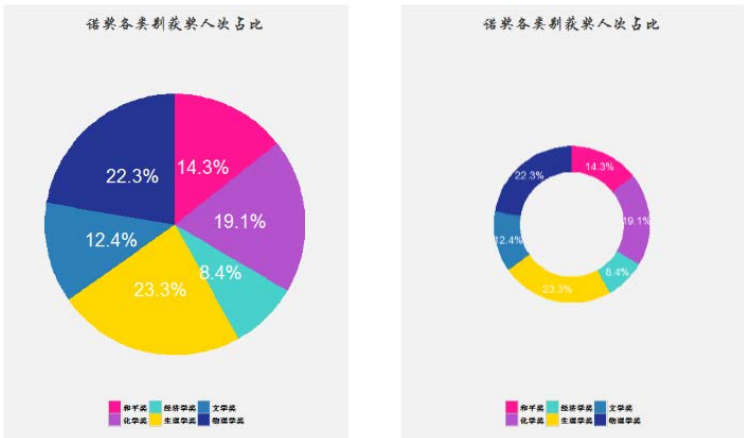


图 3-47

玫瑰图

玫瑰图绝对可以称得上是数据可视化的里程碑之作，先说一下这种图怎么看，弧度的大小为各大类之间的差异标尺，大类下的小类差异使用半径标尺，即用 width 表示，最忌讳跨大类之间的小类比较，因为跨类别比较升级成了角度和半径的乘积之比，想要进行横向比较除非进行重新定义，至少不建议跨类和类内的连续比较。

• 玫瑰图

```
1 p7 <- as.data.frame(table(p5$share, p5$cat), stringsAsFactors = FALSE)
2 p <- ggplot(p7, aes(x = Var2, y = Freq, fill = Var1)) +
3   geom_bar(width = 1, stat = "identity") +
4   coord_polar("x", start = 0) +
5   scale_fill_manual(values = c("#FF1493", "#B452CD", "#48D1CC", "#253494")) +
6   title_with_subtitle("诺奖各类别获奖分享情况") +
7   theme_bw(18) + pietheme +
8   theme(panel.grid = element_line(colour=rgb(red = 146, green = 146, blue
= 146, max = 255),size=.75),
9     axis.text.x = element_text(),
10    axis.text.y = element_blank())
```

如果分类只有一个维度，那么南丁格尔玫瑰图和饼图几乎没有多大区别；但是如果你的分类有两个维度，差别就大了，首先要将大类变量映射为 X 轴，然后小类映射为填充值，数值型变量仍然映射为 Y 轴（大小类不是用因子多少定义的，而是根据业务定义的），这还只是映射的差别，更重要的是玫瑰图 coord\_polar 旋转的不是 Y 轴，而是 X 轴，网络上大多数的博客都搞错了。如果旋转 Y 轴，你会发现各小类别的半径大小没有变化，而南丁格尔图的精髓即使用半径和弧度的差异体现数据差异，所以理解这一点就很容易发现旋转 Y 轴的错误之处了。

效果如图 3-48 所示。



图 3-48

南丁格尔玫瑰图也有缺点，很久以前新颖的创意如今已经算是司空见惯的类型了，另外它存在视觉上的误导，即面积越大数值越大，其实半径和弧度才真正代表着尺度差异。

### 雷达图

雷达图基本上和玫瑰图一样，只不过限定了弧度的大小，也就是说弧度不代表任何数据，不存在数据映射，那种使用雷达图的面积评价数据差异的方法我极为反对，因为雷达图唯一的数据映射就是类的半径，除了依据半径的比较外，其他看图方式均有失偏颇。

#### • 雷达图

```
1 temp <- p3[length(p3[, 1]):439,]
2 temp <- temp[order(temp$cat),]
3 p <- ggplot(temp, aes(y = Freq, x = cat,
4 group = simpchinese, colour = simpchinese))+
5   geom_polygon(fill = NA, size = 1) +
6   coord_polar() +
7   scale_color_manual(values = c("#FF1493", "#B452CD", "#48D1CC")) +
8   annotate("text", x = c(5, 5, 5), y = c(20, 40, 60), colour = "#008B8B",
9 label = c(20, 40, 60), size = 4, fontface = "bold", hjust = 0) +
10  title_with_subtitle("美英德三国各类别获奖情况", "文理强弱") +
11  theme_bw(18) + pietheme +
12  theme(panel.grid = element_line(colour=rgb(red = 146, green = 146, blue
13 = 146, max = 255),size=.75),
14        axis.text.x = element_text(),
15        axis.text.y = element_blank())
```

使用 ggplot2 绘制雷达图着实要走点弯路，我们选择获得诺贝尔奖最多的美、英、德三国分析：首先数据要按照即将映射到 X 轴的奖项列（cat）排序；然后将数值变量 Freq 映射到 Y 轴，对比类别 cat 映射到 X 轴，对比组 group 按照国家中文名称列分组；使用 geom\_polygon 函数添加中空的多边形图层，也是没有办法的办法，不然无法形成闭合的曲线，但基本上完成了任务。由于雷达图只有数据映射到半径，所以使用 annotate 添加文本的方式为雷达图添加了刻度，当然你也可以使用添加标签的方式完成刻度标记，大家想一想为什么玫瑰图没有添加刻度标签？因为它的半径和弧度都映射了数据，添加标签反而更乱，但是我建议最好都不添加标签，因为每个分类的尺度可能都不一样，添加无任何意义。效果如图 3-49 所示。

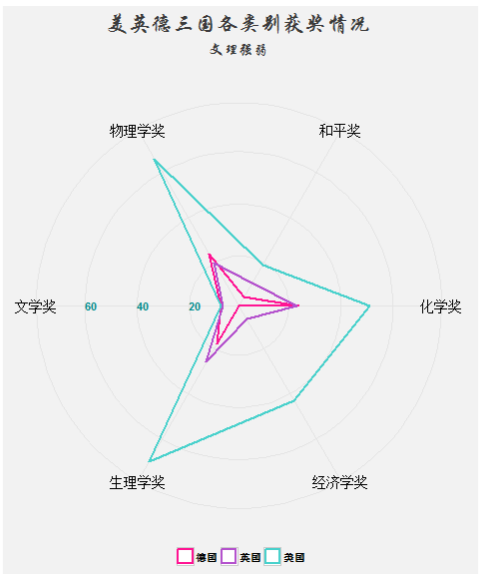


图 3-49

这类图（玫瑰图、雷达图）就是了解整体优势而已，如果用这种图深入研究数据，那真的是“图不醉人人自醉”。

### 3.2.4 古老的地图焕发新颜

地图应该是最古老的数据可视化方式，我们这里主要为地图映射上更多的数据，从地域上分析数据分布的特征，选择 2014 年《中国卫生和计划生育统计年鉴》，对全国的医疗资源做一次地域性的可视化。

#### 1. 绘制热力地图

热力图（heatmap）又称色阶图，即将数据的大小、类型映射为几何形状的颜色。绘制地图首先

## 探寻数据背后的逻辑：R 语言数据挖掘之道

需要地图文件，可以去百度搜全国或者某个省份的 SHP 文件，下载之后只需要找到 bou2\_4p.shp、bou2\_4p.shx、bou2\_4p.dbf，将三个文件放在同一目录下，读取 bou2\_4p.shp，代码如下所示。

- 读取地图数据

```
1 library(maptools)
2 mymap <- readShapePoly("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气
质的数据可视化说起/data/map/bou2_4p.shp")
3 temp <- mymap@data
4 xs <- data.frame(temp,id = seq(0, 923, 1))
5 library(ggplot2)
6 mymapd <- fortify(mymap)
```

上述代码中，第 2 行代码使用 maptools 包的 readShapePoly 函数读取地图 SHP 文件，bou 代表 boundary 的意思，数字 1~4 代表国家、省、市、县，我们读取了省级地图；第 3 行代码提取省份相关的信息赋值给 temp 对象，这里我们使用 @ 符号提取，这是一个新的提取符号，用于提取 S4 对象的元素，这里我们学习了 \$、@、[]、[[ ] ] 等四种提取符号；第 4 行代码给省份编排 id，id 从 0 开始，编排到 923 号，即每个省份模块都有一个 id，temp 所有省份的模块加一起共 924 个；地区文件有其固有的数据类型，而我们要使用 ggplot2 绘制地图自然需要将其转化为常用的数据框格式文件，第 5 行和第 6 行代码来自 ggplot2 包的 fortify 函数将 SHP 整理为数据框格式。

- 读取卫生年鉴数据

```
1 healthdata <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气
质的数据可视化说起/data/map/bou2_4p.shp", header = T, sep = ",", stringsAsFactors
= F)
2 healthdata$各地药费用 <- healthdata$各地药费用/100000000
3 mydata <- plyr::join(healthdata[, c("NAME", "各地药费用")], xs[, c("NAME",
"id")], type = "full")
4 temp <- data.frame(NAME = healthdata$NAME, lat = healthdata$lat, long =
healthdata$long)
```

上述代码中，第 1 行代码读取卫生资源数据；第 2 行代码将医药费用的单位转化为亿元；第 3 行代码将数据框 healthdata 的 NAME、各地药费用与 xs 的 NAME、id 取并集关联成为一个新的数据框 mydata；第 4 行代码去除省份省会的经纬度坐标和省份名称，用于标注地图上的省份标签。

- 绘制全国医药费用热力图

```
1 p <- ggplot(mydata) + geom_map(aes(map_id = id, fill = 各地药费用), color
= "white", map = mymapd)
2 p <- p + geom_point(data = temp, aes(x = long, y = lat, fill = NULL), colour
= rgb(red = 165, green = 165, blue = 165, max = 255)) library(directlabels)
3 p <- p + geom_dl(data = temp, aes(x = long, y = lat, label = NAME), method
= list('last.points', cex = 0.6, hjust = 1))
4 p <- p + scale_fill_gradient(name="医药费用\n(亿元)", high = rgb(red = 254,
green = 67, blue = 101, max = 255), low = rgb(red = 162, green = 162, blue = 145,
max = 255), breaks = c(5.4, 103.5, 153.3, 202.9, 317.3, 581.1))
5 p <- p + expand_limits(x = c(73, 136), y = c(6, 54))
```



```

6 library(mapproj)
7 library(maps)
8 p <- p + coord_map()
9 p

```

上述代码中，第1行代码开启画板，将id映射为地图形状的map\_id，各地医药费用用于填充地图形状的颜色，形状的边界颜色设置为白色，省份的边缘线设置为白色，map用于设置数据源，这时我们还看不到地图；第2行代码给上一步生成的画板添加省会标注点，使用temp数据框的long、lat设置经纬度分别对应X轴和Y轴，使用rgb色系设置标点颜色，到这里在终端输入p可以查看到地图；第3行代码加载directlabels，这个包中的geom\_dl函数能够自动调整标签的位置，避免标签重复；第4行代码的scale\_fill\_gradient函数用于填充的标度，参数name用于设定图例名称，“\n”表示换行，设定了最大值的颜色和最小值的颜色，并使用breaks设定了分类的组距，这个组距是根据医药费用的百分位数、均值设定的，你可以随意改变，这里我们地图中各省市的颜色被医药费用的色阶填充了，颜色越红费用越高，越暗淡费用越低；可能你已经注意到地图超出了我们坐标轴的限度，第5行代码使用expand\_limits函数延展X、Y轴的大小值，你是否想起了通过改变坐标轴的最大值误导受众的手段；第6行和第7行代码加载两个包；为什么中国的形状显得那么变形呢？没关系，第8行代码使用coord\_map对其调整一下，因为经纬线的划分有不同的体系，这样热力地图基本上画完了。

```

1 theme <- theme(panel.grid.minor = element_blank(),
2               panel.grid.major = element_blank(),
3               panel.background = element_rect(fill=rgb(red = 238,
4               green = 236, blue = 225, max = 255)),
5               plot.background = element_rect(fill=rgb(red = 238,
6               green = 236, blue = 225, max = 255)),
7               panel.border = element_blank(),
8               legend.background = element_rect(fill=rgb(red = 238,
9               green = 236, blue = 225, max = 255)),
10              axis.line = element_blank(),
11              axis.text.x = element_blank(),
12              axis.text.y = element_blank(),
13              axis.ticks = element_blank(),
14              axis.title.x = element_blank(),
15              axis.title.y = element_blank(),
16              plot.title = element_text(size=10))
17 p <- p + theme

```

theme函数的内容已经讲过很多次了，而且它的内容比较固定，就不用多说了，执行给p添加上你喜欢的主题风格就可以了。

## 2. 绘制县市地图

上面绘制的是全国的地图，但是如果要绘制市县级别的地图怎么办？最好的方法去找相关市县的SHP文件，这样直接按照上面的方法即可绘制，但是如果找不到怎么办？那就从全国地图的SHP

文件中抽取来实现，但是这种方法有一个缺点就是有些模块比较复杂的省份经常会出现一些奇形怪状的东西，所以我认为这是下策。

- 读取地图数据

```
1 library(maptools)
2 mymap <- readShapePoly("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气
质的数据可视化说起/data/map/BOUNT_poly.shp")
3 names(mymap)
4 temp <- data.frame(NAME99 = mymap$NAME99, ADCODE99 = mymap$ADCODE99)
5 temp[grep("郑州", temp$NAME99), ]
```

上面我们读取的地图文件比较粗糙，只到省级。上述代码中的第 2 行代码我们读取了详细到市县级地图文件；第 3 行代码查看地图数据都有哪些维度，其中 NAME99 就是区域模块的名称，ADCODE99 是国家地理信息中心定义的区域编码；第 3 行代码将名称和编码捆绑在一起；第 4 行代码查看包含“郑州”字符的区域模块，ADCODE99 由省、地市、县各两位代码组成的六位码，41 为河南省，01 为郑州市，了解了这些基础知识绘图就简单了。

- 整理郑州市的区域模块

```
1 zhengzhou <- mymap[grepl("^4101", mymap$ADCODE99),]
2 library(ggplot2)
3 mymapd <- fortify(zhengzhou)
4 mymapd <- plyr::rename(mymapd, c("long" = "x", "lat" = "y"))
```

上述代码中，第 1 行代码根据上一步查出的郑州市的编码，筛选出所有属于郑州市的区域模块；第 3 行代码将郑州市的地图文件转化为数据框；第 4 行代码使用 plyr 包的 rename 函数重命名经纬度列。

- 自制业务数据

```
1 mydata <- data.frame(id = unique(mymapd$id))
2 mydata$sample <- runif(length(mydata$id))
3 temp <- coordinates(zhengzhou)
4 temp <- as.data.frame(temp)
5 temp$names <- zhengzhou$NAME99
```

我们需要自制一些业务数据，就叫它郑州市的烧饼分布数据吧，上述代码中，第 1 行代码提取每个区域模块的 id 并去重，生成单列数据框；第 2 行代码添加随机平均抽样的数值，给 mydata 添加新列；第 3 行代码的 coordinates 函数用于计算一个区域的中心经纬度，用于标注区域名称标签；第 4 行和第 5 行给区域的中心点数据框添加区域名称。

- 绘制郑州市地图

```
1 zzmap <- ggplot(mydata) + geom_map(aes(map_id = id, fill = sample), color
= "white", map = mymapd)
2 zzmap <- zzmap + scale_fill_gradient(name="烧饼密度", high = rgb(red = 0,
green = 204, blue = 214, max = 255), low = rgb(red = 0, green = 130, blue = 137,
max = 255))
3 zzmap <- zzmap + geom_text(aes(x = V1, y = V2, label = names), data = temp)
4 zzmap <- zzmap + expand_limits(mymapd) + coord_map() + theme
```

上述代码中，第1行代码开启画板，将业务数据 mydata 作为总领全纲的数据，geom\_map 开启地图画板，指定模块 id 和填充列及地图区域数据 mymapd；第2行代码设定色阶变化区间和标度，使用 name 参数更改相应的图例名称；第3行代码添加行政区标签；第4行代码延展坐标轴限度，调节图形形状，同时使用 theme 函数设定地图的主题风格。

我不推荐这种方法，还是去找相应的地市级 SHP 文件比较靠谱。

### 3. 绘制气泡地图

本节我们完成在地图上绘制气泡图的任务，方法基本和上面的相似。

- 读取地图数据

```
1 library(maptools)
2 mymap <- readShapePoly("H:/zimeiti/探寻数据背后的逻辑：R 语言数据挖掘之道
/bookwriting/第3章从商务气质的数据可视化说起/data/map/bou2_4p.shp")
3 temp <- mymap@data
4 xs <- data.frame(temp,id = seq(0, 923, 1))
5 library(ggplot2)
6 mymapd <- fortify(mymap)
```

- 读取卫生年鉴数据

```
1 healthdata <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第3章从商务
气质的数据可视化说起/data/map/healthmap.csv", header = T, sep = ",", stringsAsFactors
= F)
2 healthdata$各地药费用 <- healthdata$各地药费用/100000000
3 mydata <- plyr::join(healthdata[, c("NAME", "各地药费用")], xs[, c("NAME",
"id")], type = "full")
4 temp <- data.frame(NAME = healthdata$NAME, lat = healthdata$lat, long =
healthdata$long, 各地药费用 = healthdata$各地药费用)
```

上述代码中，最后一行代码给 temp 添加了各地药费用列，用于映射气泡的大小。

- 绘制全国医药费用热力图

```
1 p1 <- ggplot(mymapd) + geom_polygon(aes(x = long, y = lat, group = id),
colour = rgb(red = 165, green = 165, blue = 165, max = 255), fill = NA)
2 p1 <- p1 + geom_point(data = temp, aes(x = long, y = lat, size = 各地药费用),
position = position_jitter(width=.5, height=1), colour = rgb(red = 254, green
= 67, blue = 101, max = 255), alpha = 0.8)
3 p1 <- p1 + scale_size_area(name="医药费用\n(亿元)", max_size = 25)
4 library(directlabels)
5 p1 <- p1 + geom_dl(data = temp, aes(x = long, y = lat, label = NAME), method
= list('last.points', cex = 0.6, hjust = 1))
6 p1 <- p1 + expand_limits(x = c(73, 136), y = c(6, 54))
7 library(mapproj)
8 library(maps)
9 p1 <- p1 + coord_map()
10 p1 <- p1 + theme
```

上述代码中,第 1 行代码开启画板,指定数据源,这次我们将 `geom_map` 函数换成了 `geom_polygon`,后者是 `ggplot2` 中用于绘制多边形的函数,而地图本质上就是多边形,这里我们将填充 `fill` 设置为 `NA`,改变了省份边界的颜色;第 2 行代码添加气泡, `position_jitter` 能够将气泡(形状)在原来位置的基础上随机移动一下,以减轻重叠现象,我们设置了气泡的颜色和透明度;第 3 行代码设置气泡大小的标尺,设置最大的气泡大小为 25;第 5 行代码设置省份的标签;第 6 行代码延展坐标轴;第 9 行代码调节图形;第 10 行代码设置主题风格。

### 4. 多图组合

本小节内容在做项目时很少用到,如果是绘制一个图版没必要这么做,你只需要将所有图的背景设为透明,然后在 PPT 里面想怎么组合就怎么组合。

其实这叫作一版多图, `ggplot2` 里面的 `facet` 实现了特殊情况下的一版多图,我们这里提供另外一套方法,如下所示。

- 绘制多图

```
1 p
2 p1
3 library(cowplot)
4 plot_grid(p, p1, align = "v")
```

我们将上面生成的地图捆绑为一版多图,使用 `cowplot` 的 `plot_grid` 函数将图捆绑在一起,当然它还有很多可以设定的参数,另外多图还可以参考 `grid.arrange` 函数。

## 3.3 将静态图转为 D3 交互图表: plotly

随着分析报告不断地被 BI 和数据挖掘的项目固化,原来的静态图表逐渐被 D3 类型的互动图表替换,数据可视化真的不仅仅限于图表,有些可视化作品做成视频更加壮观,这里我们就不再展示 GIF 动图的制作了,但是 D3 图表是无论如何也无法越过不讲的内容。R 相关的 D3 类图表有 `plotly` 和 `Echart`, `plotly` 的核心维护人员比较少,显得各方面不怎么高端大气,但是它是和 R 结合得最好的互动图表,更重要的是它能将绝大多数的 `ggplot2` 包绘制的图表转化为互动图表,而 `Echart` 是由百度的团队开发的,高端大气上档次,但是还没有 R 包灵活。

- 将 `ggplot2` 静态图转为互动图表

```
1 library(plotly)
2 library(ggplot2)
3 ggplotly(zzmap)
4 p <- ggplot(data = diamonds, aes(x = carat, y = price)) +
  geom_point(aes(text = paste("Clarity:", clarity)), size = 2) +
  geom_smooth(aes(colour = cut, fill = cut)) + facet_wrap(~ cut)
5 (gg <- ggplotly(p))
```

上述代码中，第1行和第2行代码加载 plotly 包和 ggplot2 包；第3行代码使用 ggplotly 函数将我们3.2节绘制的郑州市地图转化为交互图表；第4行代码使用 diamonds 数据绘制一张新的图；最后一行代码将刚刚绘制的图表转化为交互图表，在代码的外面使用英文小括号()表示将代码执行结果展示出来。

其他例子我们不再累举，这绝对是一个让人耳目一新的功能，救活了即将被淘汰的 ggplot2, plotly 可以将 ggplot2 融入 BI 中，让成为鸡肋的 shiny 包也会跟着神气一把。

## 3.4 从基础到进阶的变形图表

本节我们重点学习一些变形图。图表设计的最高境界，就是发挥自己的想象力设计出基于基础图表的变形图，给人带来扑面而来的春风的味道，就是“新颖”。

### 3.4.1 马赛克图（分类变量描述性分析）

马赛克图是饼图的变形图，用于绘制分类过多的情况，它将饼图的分析维度拓展到多维度，我们不妨分析下不同级别不同等次的医院在医疗市场中的份额。

- 马赛克图

```
1 hospital <- read.csv("H:/zimeiti/探寻数据背后的逻辑：R 语言数据挖掘之道
/bookwriting/第3章从商务气质的数据可视化说起/data/hospital.csv", header = T, sep =
",", stringsAsFactors = F)
2 temp <- aggregate(总收入 ~ 医院级别 + 医院等次, data = hospital, FUN = sum)
3 temp <- reshape2::dcast(data = temp, formula = 医院级别 ~ 医院等次, value.var
= "总收入", fun.aggregate = sum)
4 row.names(temp) <- temp[, 1]
5 temp <- temp[, -1]
6 library(vcd)
7 mosaicplot(temp, main = "级别 VS 等次", xlab = "级别", ylab = "等次", col=c(11,
13, 6))
```

上述代码中，第1行代码读取医疗数据；第2行代码将总收入按照医院级别、医院等次分组汇总；第3行代码使用 reshape2 包的 dcast 函数将长表变宽表；绘制马赛克图需要将数据框转化为 table 结构，也就是要将第1列的分类变量用来命名数据框的行，然后将第1列删除，这就是第4行和第5行代码完成的任务；第6行代码加载绘制马赛克图的包；第7行代码使用 mosaicplot 函数绘制马赛克图，第1个参数指定一个数据表，这里设定为 temp，temp 的行代表医院级别，被映射为 X 轴，temp 的列代表医院等级，被映射为 Y 轴，参数 col 用于图形颜色，其实使用 ggplot2 的 geom\_rect 函数也能实现，只不过比较烦琐罢了。

从图 3-50 可以看到三级甲等、二级甲等医院占据了医疗市场的绝大部分收入，这下大家应该明白了，马赛克图就是饼图的变形，只是将维度的极坐标饼图转化为了多维度的面积图。

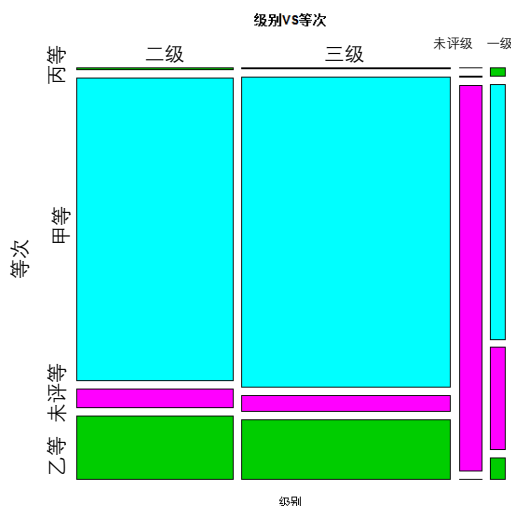


图 3-50

### 3.4.2 Sankey 图和 chordDiagram 图

还记得我们前言中讲了数据挖掘的流程图吗？那张图就是 Sankey 图，它本质上是社交网络关系图，只不过将其中的元素分了等级，最早的原型应该是拿破仑讨伐莫斯科的那张如同河流一样的图。

从图中，我们可以观察到绘制这张图至少需要以下数据：首先表示关系从 A~B 的数据，并且每个关系都有权重（value），这就是我们的第 1 张表 edges 表，用于描述上图关系及关系大小；然后我们需要标注节点的等级，比如“数据挖掘”属于一级节点，“问题边界”属于二级节点，这个节点的等级用来标注它们在 X 轴的位置，我们共分为 5 个等级，如 nodes 表中的 type；最后还需要一列表示分支，同一个分支标注相同的颜色，我们将一级和二级节点加一起共分 9 个分支，后面三级、四级、五级节点需要标注分支从属关系，比如我们将“问题边界”等设为第二个分支，那么从“问题边界”“问题描述”“业务交流”这一条线上的元素统统标为分支 2，这就需要新的一列数据，如 nodes 中的 m。

- 数据读取（1）

```
1 edges <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数据可视化说起/data/datamining.csv", header = T, sep = ",", stringsAsFactors = F)
2 names(edges) <- c("N1", "N2", "Value")
3 nodes <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数据可视化说起/data/dataminingtype.csv", header = T, sep = ",", stringsAsFactors = F)
4 nodes <- nodes[length(nodes[,1]):1,]
5 edges <- edges[length(edges[,1]):1,]
```

上述代码中，第 1~3 行代码分别读入了 edges 和 nodes 两个数据框，它们包含了绘图所需的所有标注数据，可以检查它们的数据列是不是我们所必需的；尽管 rCharts 也能绘制互动的 Sankey 图，但是就这个包本身还存在很多缺陷，所以我们使用 riverplot 包绘制 Sankey 图，但它绘制 Sankey 图时，在数据框中先出现的关系和节点反而在图形的底部（除非调整代码），所以我们要将原来的数据框重新颠倒一下首尾，第 3 行和第 4 行代码用于完成这项任务，把行编号最大的放在第一位，最小的放在末尾。

- 数据读取（2）

```
1 temp <- rep(1, length(nodes[,1]))
2 temp[which(nodes$type == 1)] <- 0
3 temp[which(nodes$type == 2)] <- 0:(length(temp[nodes$type == 2]) - 1)
4 temp[which(nodes$type == 3)] <- 0:(length(temp[nodes$type == 3]) - 1)
5 temp[which(nodes$type == 4)] <- 0:(length(temp[nodes$type == 4]) - 1)
6 nodes <- cbind(nodes, temp)
7 names(nodes) <- c("ID", "x", "m", "y")
```

节点的类别被映射为 X 轴，第一类对应 X 轴刻度 1 的位置；除了 X 轴以外，上述代码中还需要 Y 轴标定节点位置，比如第二类共包括 8 节点，那这 8 个节点谁在上面谁在下面呢？就需要给它们一个在 Y 轴上的刻度；上述代码中第 1 行代码生成一个 temp，节点有多少就复制多少次 1；第 2 行代码将类型为 1（就一个）相应的 temp 赋值为 0；第 3 行代码将类型为 2 的筛选出来，同时生成一个 0 开始以类型 2 的长度减 1 结束的序列，并赋值给相应的 temp；第 4 行和第 5 行代码重复相应的操作；第 6 行代码将 temp 和 nodes 捆绑在一起；第 7 行代码重命名 nodes。

- 生成 riverplot 对象

```
1 require("extrafont")
2 loadfonts(device = "win")
3 title <- "数据挖掘项目流程图"
4 require("RColorBrewer")
5 palette <- brewer.pal(9, "Set3")
6 styles <- lapply(nodes$m, function(n) {
  list(col = palette[n], lty = 0, textcol = "black")
})
7 names(styles) <- nodes$ID
8 rp <- list(nodes = nodes, edges = edges, styles = styles)
9 require("riverplot")
10 class(rp) <- c(class(rp), "riverplot")
```

上述代码中，前两行代码加载 extrafont 包，加载 Windows 的字体，备用；第 3 行代码设置图标题；第 4 行代码加载颜色画板包；第 5 行代码使用 brewer.pal 在画板 Set3 中取出 9 种颜色；我们要将统一分支标注为同一种颜色，上面已经说了 nodes 数据框中的 m 标志着节点是否是统一分支，统一分支 m 列的数字相同，所以针对每一个节点我们使用 m 列的数字赋予相同的颜色值，这里使用了 lapply 函数将 nodes 的 m 每一个数值传递给匿名函数，匿名函数生成一个 list，list 包含 3 个内容，从画板 palette 按 m 值提取颜色，lty 线性类型和节点标签文字的颜色，这些 list 组成一个更大的 list

返回并赋值给 styles；第 7 行代码给 styles 的元素命名，使用 nodes 的 ID 列，即节点名称；第 8 行代码将 nodes、edges、styles 捆绑为一个更大的 list，这个 list 包含了转化为 riverplot 对象的所有数据；第 9 行代码加载 riverplot 包；第 10 行代码将 rp 的类转化为 riverplot 类，这样数据就准备完成了。

## • 绘制 Sankey 图

```
1 require("grid")
2 png("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 3 章从商务气质的数据可视化说起/plot/数据挖掘流程图.png", width = 1300, height = 700, res = 1000)
3 op <- par(cex = 0.8, bg = rgb(242, 242, 242, max = 255))
4 plot(rp, plot_area = 0.85, yscale = 0.06, srt = 0)
5 grid.text(x = 0.55, y = 0.96, label = title, gp = gpar(fontfamily = "STXingkai", cex = 1.5, bg = "red"))
6 grid.text(x = 0.13, y = 0.12, label = "条带宽度：项目时间占比", gp = gpar(fontfamily = "Microsoft YaHei", cex = 0.7))
7 grid.text(x = 0.85, y = 0.06, "大音如霜工作室", gp = gpar(fontfamily = "STXingkai", cex = 1))
8 par(op)
9 dev.off()
```

上述代码中，第 1 行代码加载 grid 包，因为绘图时要用到 grid.text 函数；第 2 行代码开启 png 设备，设置图片的宽、高、像素（res）；第 3 行代码使用 par 函数设置字体大小和背景颜色，par 是基础绘图包中很重要的函数，其地位相当于 ggplot2 中的 theme 函数；第 4 行代码绘制 Sankey 图，第 1 个参数指定数据源，即我们刚刚准备好的 riverplot 的对象，plot\_area 画板面积用于绘图区的比例；yscale 设定 Y 轴的标度，srt 用于指定标签的旋转角度，这里指定不旋转；第 5 行代码的 grid.text 用于给图表添加文字，这里添加标题，x、y 参数指定标题位置，label 参数指定标题内容，gp 设定和标题相关的内容，可以设定字体类型、大小和背景色；第 6 行代码添加注释；第 7 行代码添加 LOGO；第 8 行代码设定图形风格；第 9 行代码关闭设备，这样我们的 Sankey 图就绘制完成了。效果如图 3-51 所示。

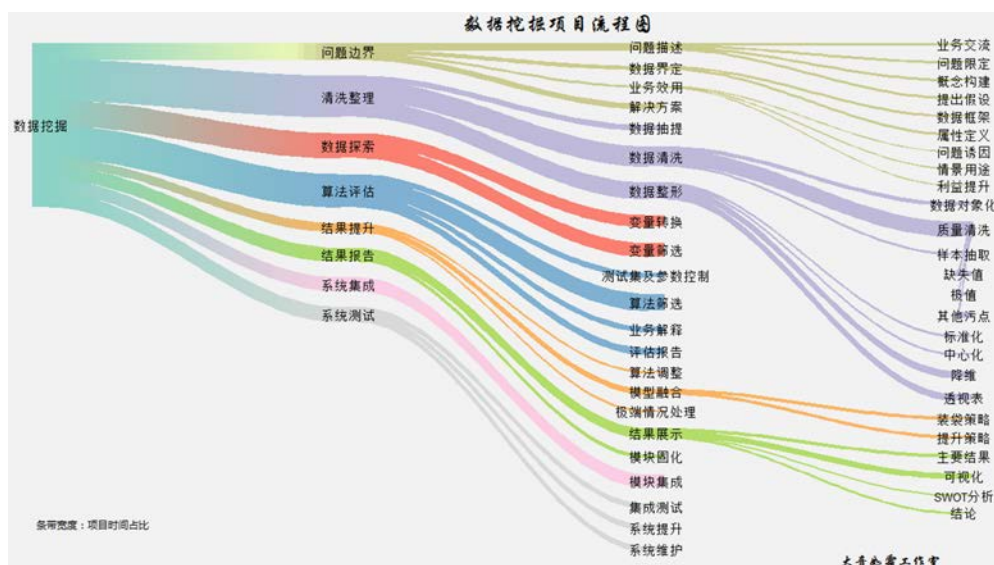


图 3-51



Sankey 图还有一种变形，就是将节点围成一个圆形的结构，俗称 chordDiagram 图，这里我们使用全球各个语言的学习成本绘制一幅 chordDiagram 图。我们将全球语言的学习成本分为 4 类：“575~600 小时”“750~900 小时”“1100 小时”“2200 小时以上”。

- 数据准备

[illegible]

上述代码中,第1行代码生成了一个全球85种语言的与它们对应的学习成本的矩阵,1、2、3、4表示它们属于哪一类;第2行代码命名矩阵的列;第3行代码命名矩阵的行;第4行代码查看矩阵。

- 绘制 chordDiagram 图

```
1 library(circlize)
2 matp2 <- prop.table(mat, margin = 2)
3 m <- matrix(1:4, 2)
4 prop.table(m, 1)
5 prop.table(m, 2)
6 chordDiagram(matp2, annotationTrack = "grid", preAllocateTracks =
list(track.height = 0.3))
```

上述代码中，第 1 行代码加载包 `circlize`；第 2 行代码计算表的百分比，这个函数很有意思，它按照指定要求计算每一个 cell 占据行总和的比例或者列总和的比例，计算前者 `margin` 指定为 1，后者指定为 2，这里指定为 2；第 3~5 行代码示例；第 6 行代码使用 `chordDiagram` 函数绘制 `chordDiagram` 图，第 1 个参数指定数据源，参数 `annotationTrack` 用于设定边缘注释的类型，如果设置为 `name`，则

边缘直接显示节点名称，如果设置为 grid，将在边缘使用色块标记注释（track），你可以分别设置，看一下区别，preAllocateTracks 用于设定 track 占单位圆的百分比，百分比越大图越小。

- 设定主题风格

```
1 circos.trackPlotRegion(track.index = 1, panel.fun = function(x,y) {
2   xlim = get.cell.meta.data("xlim")
3   ylim = get.cell.meta.data("ylim")
4   sector.name = get.cell.meta.data("sector.index")
5   circos.text(mean(xlim), ylim[1], sector.name,facing = "clockwise",
6               niceFacing = TRUE, adj = c(0,0.5)), bg.border=NA)
```

一个 chordDiagram 图包括两个部分：注释（track）和连接（link），注释包括 3 类：name、grid、axis，这里修饰 name，就是标签。上述代码中，circos.trackPlotRegion 修改注释的一些风格，参数 track.index 设定注释的编号，表示要修改哪一类注释，panel.fun 设置注释要做哪些改变，首先设定了注释的 X、Y 轴的最大值和最小值，提取了 sector.index 作为 sector.name，也就是标签，circos.text 用于将文字制成一个圆形，可以设置标签的位置、方向（sector.name,facing）、niceFacing 调节文字以适应人眼阅读，最后 bg.border 将背景边缘设置为空。效果如图 3-52 所示。

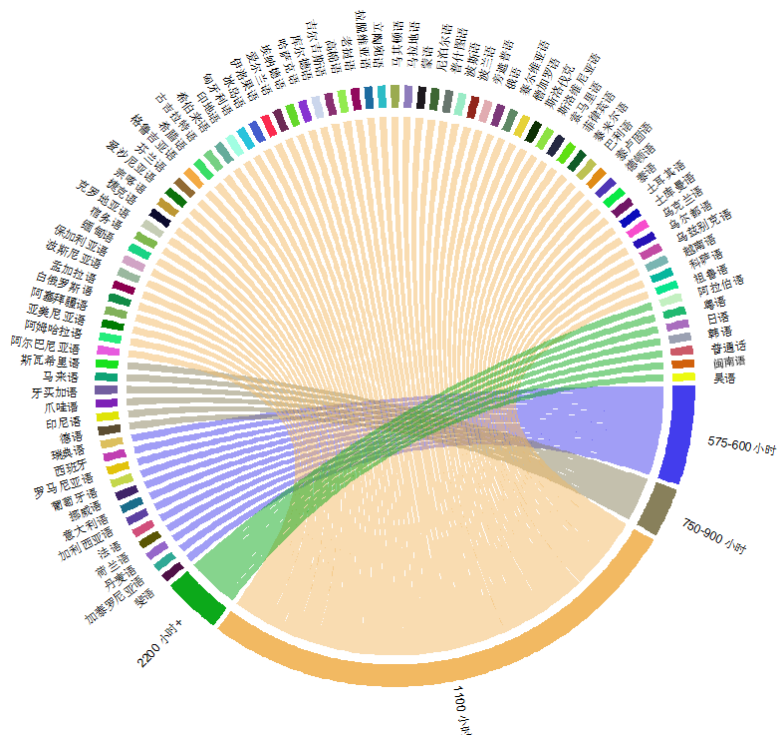


图 3-52

这样，我们的 chordDiagram 图就绘制完成了，可以看到世界上难学的语言几乎全部在中国。

# 第 4 章

## 分位数回归模拟股票指数风险通道

回归分析应该是最常见的也是最简单的统计分析方法之一，本章我们先解决一些常见的回归拟合问题，然后尝试使用分位数回归（一种与众不同的回归方法）分析上证指数，主要用于熟练掌握 R 语言中模型构建的方法。

### 4.1 用线性回归预测医院的药品销售额

在现实分析工作中，我们经常碰到先知道变量  $x$ （自变量），然后知道变量  $y$ （因变量）的情况，为了未卜先知、先人一步，我们需要使用  $x$  预测  $y$ ，这时就可以使用回归分析解决上述问题，比如我们先从单变量线性回归入手，使用医院的病床数据构建模型预测医院的总收入，其一般式如下：

$$y = bx + c$$

我们只需要使用构建模型求取  $b$  和  $c$  就可以了，如下所示。

#### • 数据探索

```
1 hospital <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 4 章分位数回归  
模拟股票指数风险通道/data/hospital.csv", header = T, sep = ",", stringsAsFactors = F)  
2 summary(hospital)  
#      省份      地级城市      单位名称  
# Length:19967      Length:19967      Length:19967  
# Class :character      Class :character      Class :character  
# Mode :character      Mode :character      Mode :character  
#      医院级别      医院等次      床位数  
# Length:19967      Length:19967      Min. : 0.0  
# Class :character      Class :character      1st Qu.: 30.0  
# Mode :character      Mode :character      Median : 70.0  
#                                     Mean : 152.2  
#                                     3rd Qu.: 178.0  
#                                     Max. : 4029.0
```

```
#
#      诊疗人次      总收入
# Min.   :    0 Min.   :    0
# 1st Qu.: 9390 1st Qu.:   800
# Median :26494 Median :  3586
# Mean   :91436 Mean   :29413
# 3rd Qu.:83283 3rd Qu.:17354
# Max.   :4225123 Max.   :2121396
# NA's   :475
3 hospital <- hospital[!is.na(hospital$床位数), ]
4 summary(hospital)
5 plot(hospital$床位数, hospital$总收入)
```

上述代码中，读入数据之后首先使用 `summary` 函数对数据大概了解一下，除了检查是否将数据正确读入以外（比如是否存在将数值型读为字符型），还发现床位数有 796 个缺失值，总收入未存在缺失值。我们可以使用第 2 章中讲解的处理缺失值的方法填充缺失值，也可以根据数据量直接移除缺失值，这里我们采用后者移除缺失值，然后使用 `plot` 函数可视化病床数和总收入的关系（见图 4-1），发现勉强可以尝试使用线性关系拟合，其实我们也可以使用相关性检验的方式来测试是否可以使用线性模型拟合。

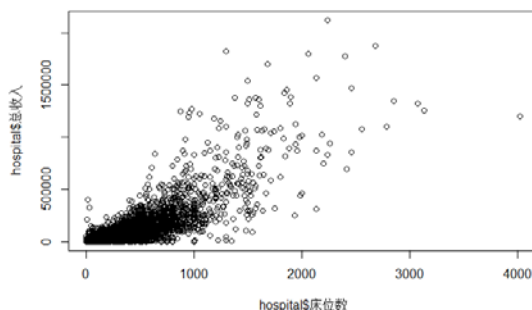


图 4-1

## ● 构建线性模型

```
1 linemodel1 <- lm(formula = 总收入 ~ 床位数, data = hospital)
2 linemodel2 <- lm(总收入 ~ 床位数 - 1, data = hospital)
3 summary(linemodel1)
# Call:
# lm(formula = 总收入 ~ 床位数, data = hospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -452189  -12228    7145   16290 1390650
#
# Coefficients:
```

```

#               Estimate Std. Error t value Pr(>|t|)
# (Intercept) -23105.599    508.828  -45.41  <2e-16 ***
# 床位数      352.663       1.851  190.48  <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 58660 on 19169 degrees of freedom
# Multiple R-squared:  0.6543, Adjusted R-squared:  0.6543
# F-statistic: 3.628e+04 on 1 and 19169 DF, p-value: < 2.2e-16
summary(linemodel2)
# Call:
# lm(formula = 总收入 ~ 床位数 - 1, data = hospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -412423  -28794  -12753   -5677  1436058
#
# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# 床位数  306.091         1.622   188.7  <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 61730 on 19170 degrees of freedom
# Multiple R-squared:  0.65, Adjusted R-squared:  0.65
# F-statistic: 3.56e+04 on 1 and 19170 DF, p-value: < 2.2e-16

```

上述代码中，第1行代码使用基础的lm函数构建线性模型，模型的第1个参数formula用于指定模型的函数关系，用“~”符号将因变量和自变量分开，前面是自变量，后面是因变量，多个因变量只需要在后面使用加号添加即可，几乎所有的监督性模型都是使用这种formula方式指定因变量和自变量之间的关系的，第2个参数指定数据集；第2行代码构建第2个一元线性模型，但是模型去掉了截距项，就是在原formula后面减去1；第3行代码使用summary函数总览构建的模型1，我们看到模型的系数和截距都非常显著，其方程可以表示为 $y = 352.663x - 23105.599$ ，模型的调整R平方为65.3%，解释度一般，调整R平方值在0和1之间，越接近1说明模型的表现越好，如果是0，则说明模型的预测结果都不如用因变量的均值进行预测；第4行代码对模型2进行了简单的摘要，模型2无截距项，但自变量床位数的参数依然非常显著，模型解释了65%的方差。

综上，线性模型的方法差强人意，因为不同类型的医院之间，收入和规模差距太大，所以如果我们先分类再构建线性模型可能效果会更好，比如我们针对不同等级的医院构建模型，当然也可以添加更多的自变量来提高模型的性能，即构建多元线性模型，代码如下所示。

- 构建多元线性模型

```

1 linemodel3 <- lm(formula = 总收入 ~ 床位数 + 诊疗人次, data = hospital)
2 summary(linemodel3)

```

```
# Call:
# lm(formula = 总收入 ~ 床位数 + 诊疗人次, data = hospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -582765  -9448    6413   13869  974190
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# (Intercept) -2.010e+04  3.978e+02  -50.54  <2e-16 ***
# 床位数       1.678e+02  2.191e+00   76.57  <2e-16 ***
# 诊疗人次     2.714e-01  2.417e-03  112.28  <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 45590 on 19044 degrees of freedom
# (124 observations deleted due to missingness)
# Multiple R-squared:  0.7924, Adjusted R-squared:  0.7924
# F-statistic: 3.634e+04 on 2 and 19044 DF, p-value: < 2.2e-16
```

上面我们构建了一个二元一次的线性模型，使用床位数、诊疗人次预测医院的总收入，而且参数的统计特征非常显著，模型的调整 R 平方上升到了 79.24%，模型性能得到了巨大的提升。

## • 分类别建模

```
1 secondhospital <- hospital[hospital$医院级别 == "二级", ]
2 linemodel2 <- lm(总收入 ~ 床位数 - 1, data = secondhospital)
3 summary(linemodel2)
# Call:
# lm(formula = 总收入 ~ 床位数 - 1, data = secondhospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -173132  -18029  -10148   -2418   560788
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# 床位数  186.073        1.655   112.4  <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 36300 on 6725 degrees of freedom
# Multiple R-squared:  0.6528, Adjusted R-squared:  0.6527
# F-statistic: 1.264e+04 on 1 and 6725 DF, p-value: < 2.2e-16
4 plot(secondhospital$床位数, secondhospital$总收入)
5 abline(linemodel2, col = "red")
```

上述代码中，我们将二级医院取出单独构建线性模型，发现模型的解释度为 65% 左右，并没有重大提升，有些让人失望；第 4 行和第 5 行代码绘制散点图和拟合线，不难发现即使医院的等级和病床数相似，收入也会存在千差万别，所以更加多样的自变量加入进来才是有效的优化方案。效果如图 4-2 所示。

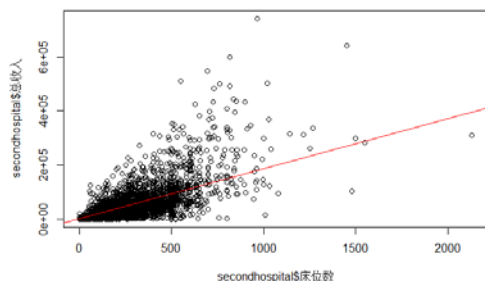


图 4-2

### 简单回归诊断

在未经测试的数据或新数据检验之前，我们可以通过回归模型的摘要和诊断对模型的效果深入研究，但是比较遗憾的是，回归诊断并不能帮助我们了解模型在面对新数据时的表现，比如是否过拟合等，我们后面将构建一些指标评价模型的效果，这里我们可以通过四幅图对回归模型在训练数据上的表现做出诊断，代码如下。

- 回归诊断

```
1 par(mfrow = c(2, 2))
2 plot(linemodel2)
```

上述代码中，第 1 行代码表示生成一个  $2 \times 2$  的图板，共绘制 4 幅图，如图 4-3 所示，分别是 Residuals vs Fitted 图、Normal Q-Q 图、Scale-Location 图、Residuals vs Leverage 图。

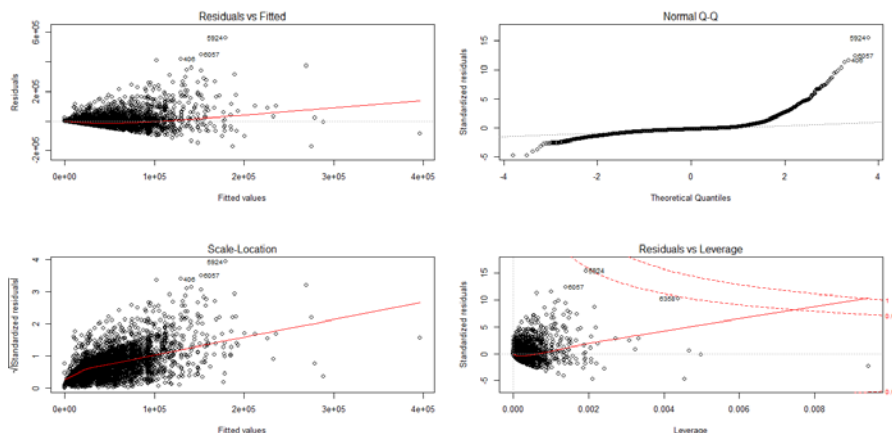


图 4-3

Residuals vs Fitted 图用于分析残差和预测值之间的趋势关系，其目的是分析因变量和自变量之间除模型所描述的关系外是否存在其他关系。若因变量与自变量线性相关，那么残差和预测值之间的散点图不会表现出任何趋势关系，因为系统方差已经被模型解释得差不多了，换句话说，残差就是白噪声。我们可以看出模型 2 的残差和预测值还是存在一个十分模糊的指数关系或者二次关系的，模型拟合得还不错，如果不放心，你可以在模型中加入这种关系或者直接使用相关模型进行再拟合优化，同样也可以看出一些异常值的编号。

当自变量值固定时，因变量呈正态分布，则残差值也应该是一个均值为 0 的正态分布。Normal Q-Q 图就是对残差正态分布的直观描述。若满足正态假设，那么图上的点应该落在呈 45° 角的直线上；若不是如此，就违反了正态性的假设，我们可以看到有一些异常值拖累了曲线，也许使用指数转化一下会使模型的效果有所提升。

若满足等方差假设，那么在位置尺度图（Scale-Location 图）中，标准残差点应该随机分布在水平线的周围，点越散漫越符合等方差假设，如果点的分布形成某种趋势，你就要重新考虑下模型了。

Residuals vs Leverage 图用于分析单个观测点对模型的影响。并非所有的极值、异常值都能对线性回归的效果产生影响，换句话说，无论数据中某些特殊点你是否移除了，都有可能无法对回归线产生任何影响；而有些点即便不是极端值也会对你的回归线产生不可忽视的影响，因为脱离群体并不一定发生在群体的两端，而极端值并不一定和群体的趋势相反，如果它在群体的趋势中，就不会对线性回归产生影响。Residuals vs Leverage 图使用 cook 距离分析这种脱离趋势的点，一般认为 cook 距离大于 0.5 即可判断其为强影响点，如果我们将那些在 cook 距离线之外的点移除，再重新建模，模型的斜率、R 平方可能都会得到改善。

## 4.2 多项式回归及常见回归方程的书写

当然，并非所有的数据都是可以用线性模型漂亮地拟合的，有时还需要使用多项式回归，比如可能  $y = bx^2 + c$  这样的关系，我们就要用到高次模型方式。R 中构建高次模型的方法基本上和线性模型方法差别不大，如下所示。

### • 构建多项式回归模型

```
1 linemodel4 <- lm(formula = 总收入 ~ I(床位数^2), data = hospital)
2 summary(linemodel4)
# Call:
# lm(formula = 总收入 ~ I(床位数^2), data = hospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -2695189  -12259   -10521   -3148  1409331
#
# Coefficients:
```



```

#           Estimate Std. Error t value Pr(>|t|)
# (Intercept) 1.252e+04 4.458e+02 28.08 <2e-16 ***
# I(床位数^2) 2.391e-01 1.307e-03 182.96 <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 60200 on 19169 degrees of freedom
# Multiple R-squared:  0.6359, Adjusted R-squared:  0.6359
# F-statistic: 3.347e+04 on 1 and 19169 DF, p-value: < 2.2e-16
3 linemodel5 <- lm(formula = 总收入 ~ I(床位数^2) + 床位数:诊疗人次, data =
hospital)
4 summary(linemodel5)
# Call:
# lm(formula = 总收入 ~ I(床位数^2) + 床位数:诊疗人次, data = hospital)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -2385658  -12930  -10740   -1676  1061309
#
# Coefficients:
#           Estimate Std. Error t value Pr(>|t|)
# (Intercept)  1.352e+04 3.562e+02  37.96 <2e-16 ***
# I(床位数^2)   4.376e-02 2.121e-03  20.63 <2e-16 ***
# 床位数:诊疗人次 2.777e-04 2.627e-06 105.72 <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# Residual standard error: 47930 on 19044 degrees of freedom
# (124 observations deleted due to missingness)
# Multiple R-squared:  0.7706, Adjusted R-squared:  0.7706
# F-statistic: 3.198e+04 on 2 and 19044 DF, p-value: < 2.2e-16

```

上述代码中, 我们使用床位数的二次方构建模型, I函数表示对变量进行一次数学转化; 第2行代码的摘要表示尽管可以使用床位数的二次方构建模型(系数统计上非常显著), 但是模型的解释度反而下降了, 调整R平方仅为63%; 当然我们也可以构建多变量的多项回归模型, 用于研究变量交互作用、对因变量的影响, 比如构建形如 $y = ax^2 + bx:m + c$ 这类一般式的模型, 我们使用了床位数的二次方和床位数与诊疗人次的相互作用预测因变量, 从摘要中我们看到床位数、诊疗人次相互作用对自变量的影响在统计上非常显著, 但是整体上来说模型与多元一次回归模型比较模型的解释度还是有所下降, lm中用于构建formula的参数总结如表4-1所示。

表 4-1

符 号	用 途
$\sim$	等式分隔符，左边为因变量，右边自变量。如：当自变量为 $x$ 、 $m$ 、 $n$ ，因变量为 $y$ 时，代码为 $y \sim x + m + n$
$+$	分隔自变量
$:$	表示自变量的交互项。如：要通过 $x$ 、 $m$ 及 $x$ 与 $m$ 的交互项预测 $y$ ，代码为 $y \sim x + m + x:m$
$*$	表示所有可能交互项的简洁方式。如： $y \sim x * m * n$ 可展开为 $y \sim x + m + n + x:m + x:n + m:n + x:m:n$
$^$	表示交互项达到某个次数。如： $y \sim (x + m + n)^2$ 可展开为 $y \sim x + m + n + x:m + x:n + m:n$
$.$	表示包含除因变量外的所有变量。如：若一个数据框包含变量 $x$ 、 $y$ 、 $m$ 和 $n$ ，代码 $y \sim .$ 可展开为 $y \sim x + m + n$
$-$	减号，表示从等式中移除某个变量。如： $y \sim (x + m + n)^2 - x:n$ 可展开为 $y \sim x + m + n + x:m + m:n$
$-1$	删除截距项。表达式 $y \sim x - 1$ 拟合 $y$ 在 $x$ 上的回归，强制直线通过原点
$I()$	从算术的角度来解释括号中的元素。如： $y \sim x + (m + n)^2$ 将展开为 $y \sim x + m + n + m:n$ 。相反，代码 $y \sim x + I((m + n)^2)$ 将展开为 $y \sim x + h$ ， $h$ 是一个由 $m$ 和 $n$ 的平方和创建的新变量
function	可以在表达式中用的数学函数。如： $\log(y) \sim x + m + n$ 表示通过 $x$ 、 $m$ 和 $n$ 来预测 $\log(y)$

仔细阅读表 4-1，大家基本上可以完成常见的回归 formula 书写，包括高次回归、多变量回归、交互作用、对数变换等。

### 4.3 Lasso 回归和回归评价的常见指标

根据我多年阅读市场研究报告的经验，大多数回归被人用来制造趋势，特别是 Lasso 回归，被别有用心的的人在数据的散点分布图上制造一些或有或无或无法解释的趋势，这些趋势的一个特点就是制造者对他们的解读非常苍白和肤浅，与它所用的高深分析技术极不匹配，我厌恶这种图和制造这种图的人，希望大家能够根据数据和业务实事求是地讲话。

Lasso 回归能够在完成参数估计的同时实现自变量的筛选，比较优秀地解决了回归分析中多重共线性的问题，而且结果的可解释性也比较强。不论因变量是连续的还是离散的，Lasso 回归都可以建模。Lasso 的变量筛选是指不把所有的变量都放入模型中进行拟合，而是选择性地将变量放入模型从而得到更好的性能参数。复杂度调整是指通过一系列参数控制模型的复杂度，从而避免过拟合，请注意能避免过拟合不代表不会发生过拟合。

• 数据整理

```
1 lassohospital <- hospital[, c("床位数", "诊疗人次", "总收入")]
2 names(lassohospital) <- c("bed", "patient", "income")
3 lassohospital <- lassohospital[complete.cases(lassohospital),]
4 lassohospital$bedlog <- log(lassohospital$bed + 1)
```

```

5 lassohospital$patientlog <- log(lassohospital$patient + 1)
6 lassohospital$patientexp <- (lassohospital$patient)^2
7 lassohospital$bedexp <- (lassohospital$bed)^2
8 test.idx <- sample(1:nrow(lassohospital), 300)
9 hospital.test <- lassohospital[test.idx, ]
10 hospital.train <- lassohospital[-test.idx, ]

```

上述代码中,第1行代码提取我们需要的变量;第2行代码重命名;第3行代码使用 complete.cases 函数删除含有缺失值的行,只要某行含有一个缺失值,就会被移除;第4~7行代码使用对数和指数为我们多添加几个自变量(变量太少了,我们自己构造一些);第8行代码随机抽取300个行编号;第9行代码按照行编号抽取数据作为测试数据;第10行代码移除测试行编号把剩下的数据作为训练数据。

#### • 模型测试

```

1 library(glmnet)
2 modellasso <- cv.glmnet(x = as.matrix(subset(hospital.train, select =
-income)), y = hospital.train$income, family="gaussian", nfolds = 10)
3 modeltemp <- glmnet(x = as.matrix(subset(hospital.train, select =
-income)), y = hospital.train$income, family="gaussian", lambda = 500, alpha =
1, standardize = FALSE)

```

上述代码中, cv.glmnet 函数能够根据数据和 lambda 控制建立尽量多的模型,且能够自动完成交叉检验,可以通过 nfolds 设置阶数,这里我们设置了10阶交叉检验,另外要求输入的数据不能包含 NA 值,必须以矩阵而不是数据框的方式录入,所以如果要输入因子变量(离散变量),则需要将因子转化为列变量(即长表变宽表),同时用 0、1 表示值的哑变量,因为矩阵中的数据类型各列之间必须保持一致; family 参数用于指定回归模型的类型,选择何种参数根据你的因变量类型进行选择,如下所示。

#### family 参数说明

family="gaussian": 适用于一维连续因变量(univariate)

family="mgaussian": 适用于多维连续因变量(multivariate)

family="poisson": 适用于非负次数因变量(count)

family="binomial": 适用于二元离散因变量(binary)

family="multinomial": 适用于多元离散因变量(category)

根据 cv.glmnet 函数建模的结果我们可以从中筛选最优秀最简洁的模型,当然也可以直接使用 glmnet 函数设置 lambda 参数独立建模,通过参数 lambda,即  $\lambda$  值调整模型, lambda 指标用于衡量模型的复杂度,一般来说模型的变量越多 lambda 越大。 alpha 控制应对共线性数据时模型的性状, Lasso 回归 alpha=1, Ridge 回归 alpha=0, 后者是岭回归; standardize 参数用于指定是否对数据进行标准化,如下所示。

#### • 模型诊断

```

1 modellasso$glmnet.fit
# Call: glmnet(x = as.matrix(subset(hospital.train, select = -income)),

```

```
y = hospital.train$income, family = "gaussian")
#
#      Df    %Dev  Lambda
# [1,]  0 0.0000 85800.0
# [2,]  1 0.1240 78180.0
# [3,]  1 0.2269 71230.0
# [4,]  2 0.3231 64910.0
# [5,]  3 0.4086 59140.0
# [6,]  3 0.4800 53890.0
# [7,]  3 0.5392 49100.0
# [8,]  3 0.5884 44740.0
# [9,]  3 0.6292 40760.0
# [10,] 3 0.6631 37140.0
# [11,] 3 0.6912 33840.0
# [12,] 3 0.7146 30840.0
# [13,] 3 0.7339 28100.0
# [14,] 3 0.7500 25600.0
# [15,] 3 0.7634 23330.0
# [16,] 3 0.7745 21250.0
# [17,] 3 0.7837 19370.0
# [18,] 3 0.7914 17650.0
# [19,] 3 0.7977 16080.0
# [20,] 4 0.8033 14650.0
# [21,] 4 0.8081 13350.0
# [22,] 4 0.8121 12160.0
# [23,] 4 0.8154 11080.0
# [24,] 4 0.8182 10100.0
# [25,] 4 0.8205  9200.0
# [26,] 4 0.8223  8383.0
# [27,] 4 0.8239  7638.0
# [28,] 4 0.8252  6960.0
# [29,] 4 0.8263  6341.0
# [30,] 4 0.8272  5778.0
# [31,] 4 0.8280  5265.0
# [32,] 4 0.8286  4797.0
# [33,] 4 0.8291  4371.0
# [34,] 4 0.8295  3983.0
# [35,] 4 0.8299  3629.0
# [36,] 4 0.8302  3306.0
# [37,] 4 0.8304  3013.0
# [38,] 4 0.8306  2745.0
# [39,] 4 0.8308  2501.0
# [40,] 4 0.8309  2279.0
# [41,] 4 0.8311  2077.0
```

```

# [42,] 6 0.8318 1892.0
# [43,] 6 0.8326 1724.0
# [44,] 6 0.8332 1571.0
# [45,] 6 0.8337 1431.0
# [46,] 6 0.8341 1304.0
# [47,] 6 0.8345 1188.0
# [48,] 6 0.8348 1083.0
# [49,] 6 0.8350 986.5
# [50,] 6 0.8352 898.9
# [51,] 6 0.8354 819.0
# [52,] 6 0.8355 746.3
# [53,] 6 0.8356 680.0
# [54,] 6 0.8357 619.6
# [55,] 6 0.8358 564.5
# [56,] 6 0.8359 514.4
# [57,] 6 0.8359 468.7
# [58,] 6 0.8360 427.0
# [59,] 6 0.8360 389.1
# [60,] 6 0.8360 354.5
# [61,] 6 0.8361 323.0
# [62,] 6 0.8361 294.3
# [63,] 6 0.8361 268.2
# [64,] 6 0.8361 244.4
# [65,] 6 0.8361 222.7
# [66,] 6 0.8362 202.9
# [67,] 6 0.8362 184.9
# [68,] 6 0.8362 168.4

```

打印模型结果，模型在 70 个  $\lambda$  时因为 %Dev 变化很小自动中止了。数据的每一行代表一个模型，Df 列表示线性模型非零拟合系数的个数，%Dev 列表示由模型解释的方差比例，就是线性模型中模型拟合的 R 平方， $\lambda$  列是每个模型对应的  $\lambda$  值，随着  $\lambda$  变小，越来越多的自变量被接入模型，%Dev 也越来越大，即模型在训练数据上的效果越来越好。

- 选择最优模型

```

1 plot(modellasso)
2 modellasso$lambda.1se
3 modellasso.final <- modellasso$glmnet.fit
4 coef(modellasso$glmnet.fit, s = modellasso$lambda.1se)
# 7 x 1 sparse Matrix of class "dgCMatrix"
#
# (Intercept) -2.918993e+03
# bed         4.470494e+01
# patient     2.165974e-01
# bedlog      .
# patientlog  .

```

```
# patientexp 7.496769e-09
# bedexp 8.302179e-02
```

cv.glmnet 函数采用交叉检验，对不同的 lambda 值构建的模型测试了 10 次模型误差，因此我们可以绘制交叉检验图来选择最优模型。效果如图 4-4 所示。

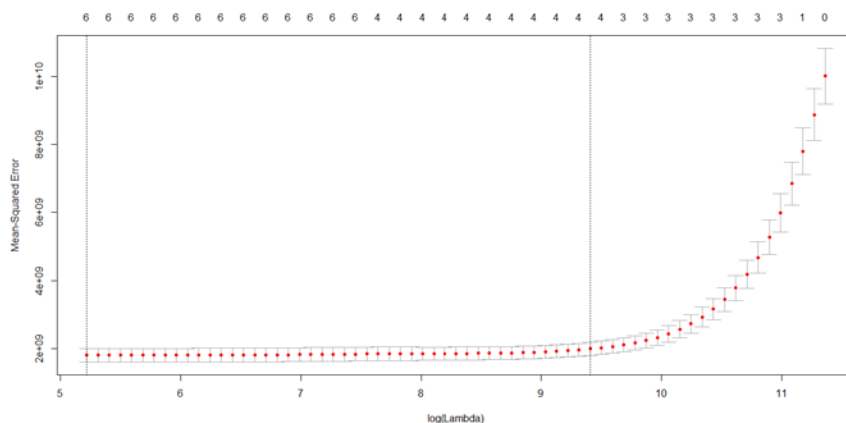


图 4-4

图 4-5 中下横轴是 lambda 值的对数，上横轴是加入模型的变量数，纵轴是均方差。最佳 lambda 取值就是曲线对应的最低点，对应模型的变量个数是 6 个；右侧虚线对应一个更加简洁的模型，一般是在曲线的拐点处，表示模型的复杂度增加并不能引起模型均方差快速下降。上述代码中，第 2 行代码可以查看最佳的 lambda 值；第 3 行代码提取最优模型；第 4 行代码按照最佳 lambda 提取最优模型的参数，可以看到模型移除了两个经过对数转换的变量。

#### • 预测结果

```
1 x <- as.matrix(subset(hospital.test, select = -income))
2 prediction <- predict(modellasso.final, newx = x, s = modellasso$lambda.
1se)
3 y = hospital.test$income
```

模型的评价除了一些统计技术方面的诊断外，就是真枪实弹用测试数据或新数据验证，因为是不是过拟合或者欠拟合需要模型在训练集和测试集上进行效果对比才能得知，当然我们也需要构建一些评价指标，这里我们构建平均绝对误差 (MAE)、均方差 (MSE)、标准化平均绝对方差 (NMSE) 3 个指标用于评价模型的效果。

#### 统计指标

- 平均绝对误差 =  $\text{mean}(\text{abs}(\text{预测值} - \text{观测值}))$
- 均方差 =  $\text{mean}((\text{预测值} - \text{观测值})^2)$
- 标准化平均方差 =  $\text{mean}((\text{预测值} - \text{观测值})^2) / \text{mean}((\text{mean}(\text{观测值}) - \text{观测值})^2)$

你能说出这些指标的优缺点吗 (参看第 10 章)? 这些应该是评价连续型目标变量比较常见的指

标，比如时间序列分析等。

- 回归模型的筛选指标

```
1 maefun <- function(pred, obs) mean(abs(pred - obs))
2 msefun <- function(pred, obs) mean((pred - obs)^2)
3 nmsefun <- function(pred, obs) mean((pred - obs)^2)/mean((mean(obs) -
obs)^2)
4 mae = maefun(pred = prediction, obs = y)
5 mse = msefun(pred = prediction, obs = y)
6 nmse = nmsefun(pred = prediction, obs = y)
7 print(c(mae, mse, nmse))
```

上述代码中，第 1~3 行代码按照定义构建了 3 个评价指标；第 4~6 行代码计算 3 个指标；第 7 行代码打印 3 个指标。

几乎所有的指标都只有在对比中才能发现优势，所谓优势都是比较优势，评价模型时，我们要进行两个维度的比较：① 测试集的效果和训练的效果比较，此比较用于分析同一类型的模型是否达到最优，比如是否发生欠拟合和过拟合，如果测试集比训练集的效果差，则有可能发生过拟合；② 不同类型的模型之间的比较，这类一般是为了筛选比较优秀的模型种类。但无论是哪种比较都要寻求统计上的意义，一般要进行交叉检验。

## 4.4 分位数回归拟合上证指数风险通道

股票市场是一个神秘的场所，大部分人拒之如洪水猛兽，但我认为一个行业存在而且能成为国家的支柱产业必然有其合理之处，所以股票市场和赌场自然有其不同之处。人的大部分时间在忙于赚钱养家，但是靠职业的发展赚钱对大多数人来说是存在“天花板”的，如果我们稍有积蓄就要想办法通过资金赚钱，正所谓“无财作力，少有斗智，既饶争时”。

作为数据分析师在入市之前你必须准备大量的知识，至少应该对大盘指数的历史数据稍加探索，比如大盘的历史均值、中位数、百分位数等，要有个大概的了解，至少要知道自己是在什么位置入场的，如果你想进一步了解市场，我建议你至少应该读一下杰里米·西格尔的《投资者的未来》。

下面我们来学习用分位数回归模拟上证股指的通道，传统的线性回归模型描述了因变量的条件均值分布受自变量的影响。其中，最小二乘法是估计回归系数的常用的基本方法。面对经济数据，这种假设通常就不满足了，比如上证指数（上海证券交易所指数）就存在大量的后尾、尖峰情况，最小二乘法的估计将不再具有优良性质。而分位数回归利用自变量的多个分位数（例如四分位、十分位、百分位等）来得到因变量的条件分布所对应的分位数方程。

使用 quantmod 包抓取最新的股票数据，如下所示。

- 加载数据包

```
1 if (!suppressWarnings(require("quantmod"))) {
2   install.packages("quantmod")
}
```

```
3   require("quantmod")
4 }
```

quantmod 包可以抓取最新的股票数据，上面的代码我们使用了一种自动的方式加载包，如果你读了本书的最后一章就会知道这种方式非常有用。

- 读取股票数据

```
1 sz <- getSymbols("000001.SS", from = "1995-01-01", src = "yahoo",
auto.assign = FALSE)
2 szdf <- data.frame(sz, exchangedate = time(sz))
3 szdf <- data.frame(szdf)
4 row.names(szdf) <- 1:length(szdf[,1])
5 tail(szdf)
#      X000001.SS.Open X000001.SS.High X000001.SS.Low X000001.SS.Close
# 5596          3113.01          3113.01          3113.01          3113.01
# 5597          3101.30          3101.30          3101.30          3101.30
# 5598          3123.14          3123.14          3123.14          3123.14
# 5599          3149.55          3149.55          3149.55          3149.55
# 5600          3159.17          3159.17          3159.17          3159.17
# 5601          3140.17          3140.17          3140.17          3140.17
#      X000001.SS.Volume X000001.SS.Adjusted exchangedate
# 5596                0          3113.01 2017-01-18
# 5597                0          3101.30 2017-01-19
# 5598                0          3123.14 2017-01-20
# 5599                0          3149.55 2017-01-25
# 5600                0          3159.17 2017-01-26
# 5601                0          3140.17 2017-02-03
6 summary(szdf[, "X000001.SS.Close"])
#  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#  516.5 1313.0 1889.0 2034.0 2630.0 6092.0
```

上述代码中，第 1 行代码使用 getSymbols 函数读取股票数据，第 1 个参数指定股票的代码，上海证券交易所（沪市）以 ss 结尾，深圳证券交易所（深市）以 sz 结尾，from 参数设定数据读取的起始时间，src 设定数据源；因为 getSymbols 函数返回的是时间序列对象，所以第 2 行代码我们使用 time 函数提取了日期数据赋值给 szdf 的新列 exchangedate；第 3 行代码将时间序列对象转化为数据框；第 4 行代码重新命名数据框的行名称；第 5 行代码使用 tail 查看一下数据最后几行，我们看到数据包括开盘价、最高价、最低价、收盘价、交易量、复权价、交易日期；第 6 行代码使用 summary 函数了解数据摘要。

对于这类时间序列数据，我们只能用历史数据预测未来，就是用今天的数据预测明天的数据，所以要将数据调整为三列，一列目标变量，即“明天”的指数值；一列自变量，即“今天”的指数值；一列日期。

- 数据调整

```
1 nameclose <- "X000001.SS.Close"
```



```

2 szdfqg <- szdf[, c("exchangedate", nameclose)]
3 closey <- szdfqg[2:length(szdfqg[,1]), nameclose]
4 closex <- szdfqg[1:(length(szdfqg[,1])-1), nameclose]
5 exchangedate <- szdfqg[2:length(szdfqg[,1]), "exchangedate"]
6 szdfqg <- data.frame(exchangedate, closey, closex)

```

因为收盘数据是多空双方激战一天的结果，所以我们仅使用收盘数据进行建模，上述代码中，第1行和第2行代码提取日期和收盘数据；第3行代码提取从第2行到最后一行数据作为目标变量closey；第4行代码提取第1行到倒数第2行数据作为自变量；第5行代码提取与目标变量对应的日期作为交易日期；第6行代码将三个变量组合为新的数据框。

- 分位数回归模型

```

1 library("quantreg")
2 fitqg <- rq(closey ~ closex, tau = c(0.05, 0.20, 0.40, 0.60, 0.80, 0.95),
data= szdfqg)
3 predict(fitqg, newdata = data.frame(closey = NA, closex = szdfqg[length
(szdfqg[,1]), "closey"]))
# tau= 0.05 tau= 0.20 tau= 0.40 tau= 0.60 tau= 0.80 tau= 0.95
# 1 3045.279 3108.911 3135.492 3151.266 3176.534 3222.326

```

上述代码中，第1行代码加载quantreg；第2行代码使用rq函数构建分位数模型，函数的第1个参数指定方程，tau指定基于什么分位数建模，我们指定了5%分位数、20%分位数、40%分位数、60%分位数、80%分位数、95%分位数构建模型，data参数指定数据集；第3行代码使用最新的数据预测明天的指数的百分位数区间，szdfqg[length(szdfqg[,1]), "closey"])提取最后一天的指数和一个空值捆绑为新的数据框作为新数据录入模型，模型返回明天指数的预测值，可以看到分位数模型返回了相应的5%分位数、20%分位数、40%分位数、60%分位数、80%分位数、95%分位数指数预测值。

- 训练数据分位数预测值

```

1 p95 <- as.data.frame(predict(fitqg))
2 names(p95) <- c(paste("p", c(0.05, 0.20, 0.40, 0.60, 0.80, 0.95), sep = ""))
3 szdfqg <- cbind(szdfqg[, c("exchangedate", "closey")], p95)
4 szdfqg <- szdfqg[(length(szdfqg$exchangedate) - 210):length
(szdfqg$exchangedate),]
5 q <- 1:length(szdfqg$exchangedate)
6 szdfqg <- cbind(szdfqg, q)
7 temp <- substr(szdfqg$exchangedate,1,7)
8 n <- q[!duplicated(temp)]
9 temp <- unique(temp)

```

我们可以使用训练数据预测一下指数值，然后绘制指数的风险通道，即将预测的百分位数作为风险直观地显示在图上，首先我们要对数据进行调整。上述代码中，第1行代码使用predict函数获得预测值，并转化为数据框；第2行代码统一重命名预测值，paste函数形成一个和数据框的列数等长的字符向量，然后命名；第3行代码将预测值与实际值、日期捆绑为新的数据框；第4行代码取最后210行数据绘图；绘图时日期数据的X轴是由q映射的，但是我们要显示日期的标签，如果全

部显示标签会出现拥挤的现象,所以我们只按年月显示;第5行和第6行代码生成和数据框等长的q,用于映射X轴;第7行代码截取日期字段的前7个字符作为标签,substr函数用于截取字符串的指定部分;但是从年月日到年月会有重复,所以第9行代码对temp(年月)去重;如果在X轴上显示刻度标签,就要有切分点,而且切分点和标签要一一对应,所以第8行代码提取与不重复的年月标签对应的q值赋值给n,这样就可以用n切分数据了。

- 加载自编绘图函数

```
1 library(ggplot2)
2 library("extrafont")
3 title_with_subtitle = function(title, subtitle = "") {
4   ggtitle(bquote(atop(. (title), atop(. (subtitle))))))
5 }
# font_import()#引入字体
6 loadfonts(device = "win")
```

上述代码加载了自编函数用于添加标题,同时加载了使用 Windows 字体的包和 Windows 下的字体,我们在第3章已经详细讲过。

- 回归股指风险通道

```
1 p <- ggplot(szdffq, aes(x = q, y = closey)) +
2   geom_line(size=1, colour = rgb(red = 253, green = 107, blue = 117, max
= 255)) +
3   geom_smooth(aes(ymin = p0.05, ymax = p0.2), data = szdffq, stat =
"identity",
4     fill = rgb(red = 229, green = 181, blue = 202, max = 255),
5     linetype=0, size=2) +
6   geom_smooth(aes(ymin = p0.2, ymax = p0.4), data = szdffq, stat =
"identity",
7     fill = rgb(red = 242, green = 218, blue = 228, max = 255),
8     linetype=0, size=2) +
9   geom_smooth(aes(ymin = p0.4, ymax = p0.6), data = szdffq, stat =
"identity",
10    fill = rgb(red = 204, green = 202, blue = 198, max = 255),
11    linetype=0, size=2) +
12   geom_smooth(aes(ymin = p0.6, ymax = p0.8), data = szdffq, stat =
"identity",
13    fill = rgb(red = 180, green = 241, blue = 242, max = 255),
14    linetype=0, size=2) +
15   geom_smooth(aes(ymin = p0.8, ymax = p0.95), data = szdffq, stat =
"identity",
16    fill = rgb(red = 123, green = 230, blue = 223, max = 255),
17    linetype=0, size=2) +
18   scale_x_continuous(minor_breaks = 0, breaks = n, labels =temp) +
19   title_with_subtitle("风险通道", "基于1995年5月至今的沪指数据建模") +
20   ylab("上证指数") +
```

```
21 xlab("时间")
```

绘图已经是我们熟悉的内容了，上述代码中，将 `closey` 和 `q` 映射为  $Y$  轴和  $X$  轴，添加了折线，为了绘制通道我们添加了阴影，使用 `geom_smooth` 函数绘制阴影，只需要指定阴影的  $Y$  轴上缘 `ymin` 和  $Y$  轴下缘 `ymin` 即可；我们使用了 `scale_x_continuous` 切分  $X$  轴标定刻度标签；其他都是我们熟悉的内容了，这里不再赘述。

- 添加主题内容

```
1 p <- p + theme_bw(18) + theme(panel.background = element_rect(fill = rgb(red
= 242,
2   green = 242, blue = 242, max = 255)),
3   plot.background = element_rect(fill = rgb(red = 242,
4   green = 242, blue = 242, max = 255)),
5   plot.title = element_text(size = rel(1.2), family = "STXingkai", face
= "bold", hjust = 0.5, colour = "#3B3B3B"),
6   panel.grid.major = element_line(colour=rgb(red = 146, green = 146, blue
= 146, max = 255),size=.75),
7   panel.border = element_rect(colour = rgb(red = 242,
8   green = 242, blue = 242, max = 255)),
9   axis.ticks = element_blank(),
10  axis.text.x = element_text(colour = "grey20", size = 12),
11  axis.text.y = element_text(colour = "grey20",
12    size = 12),
13  axis.title.y = element_text(size = 11, colour = rgb(red = 74,
14    green = 69, blue = 42, max = 255), face = "bold", vjust = 0.5),
15  axis.title.x = element_text(size = 11, colour = rgb(red = 74,
16    green = 69, blue = 42, max = 255), face = "bold", vjust = -0.5),
17  legend.position = "none")
```

以上代码只是为了给图加上统一风格的 `theme`，第3章已经讲得非常详细，这里不再赘述。

从图 4-5 中我们可以看出当指数穿过 95% 百分位数上缘不久就会有一拨下坠，当穿越 95% 百分位数下缘不久就会有一拨上行，那么我们是不是可以指定这样的策略：当指数突破上缘边线时，就卖出，突破下缘边线时就买入。这只是一个非常简单的策略。

你制定的策略只是你观察的结果，是不是能赚钱还需要回测检验，所谓回测检验就是将你的策略写成程序使用历史数据测试一下是否能赚到跑赢通胀跑赢其他投资策略的钱，毕竟谈钱无小事，都是自己的血汗钱，一定要慎重。



图 4-5

# 第 5 章

## 时间序列分析

### 5.1 时间序列分析：分析带有时间属性的数列

时间序列（time series）是指按出现时间的先后顺序排列的数据。我们最常见的最简单时间序列应该是股市中的某个股票的即时价格：在交易时段每分钟都有一个参考报价，一个交易日下来，这些价格数据就形成了一个时间序列。同样，某个股票在连续某段日子里，每个交易日的收盘价格也可以组成一个时间序列。

时间序列分析的主要作用是利用已有的数据进行探查，并对未来的数据进行预测，同时也可以用来寻找异常值等。时间序列分析是基于回归分析的思路进行的，但回归分析主要关注的是自变量与因变量之间存在的统计相关关系，时间序列分析研究的是自变量与自变量本身、自变量与时间之间的关系。

R 软件在时间序列分析方面有很多包，比较常用的有 zoo、xts、forecast 等。在这里，我们会关注时间序列分析的 3 大方面，即趋势性、周期性和波动性。

### 5.2 不是所有序列都叫时间序列

在第 4 章中曾经提到的上海证券交易所指数（上证指数）每天的收盘价格即为时间序列，我们可以用 xts 包的 xts() 函数构建一个基本的时间序列对象，代码如下。

- xts 对象绘图

```
1 library("quantmod")
2 SS <- getSymbols("000001.SS", from = "1995-01-01", src = "yahoo",
auto.assign = FALSE) ###
3 SS_close = xts(SS[, "000001.SS.Close"], order.by=time(SS))
```

```
#[1] "xts" "zoo"
4 summary(SS_close)
#Index      000001.SS.Close
#Min.   :1995-01-02   Min.    : 516.5
#1st Qu.:2000-05-15   1st Qu.:1313.2
#Median :2005-09-26   Median :1889.0
#Mean   :2005-11-02   Mean    :2034.0
#3rd Qu.:2011-03-24   3rd Qu.:2632.3
#Max.   :2017-02-06   Max.    :6092.1
5 plot.xts(SS_close)
```

效果如图 5-1 所示。



图 5-1

从技术上来说，任何一个数据都可以成为时间序列，只要给这些数据一个时间的先后顺序就行了。但其实这种数据用时间序列的方法来分析是毫无意义的。

- 随机生成时间序列

```
1 number=ts(sample(seq(1,37,1), size=100, replace=T))
2 ts(number)
#Time Series:
# Start = 1
#End = 100
#Frequency = 1
#[1] 12 21 37 19 21 10 35 19 2 22 27 4 6 19 16 37 6 29 10 8 33 33 27 35
#[25] 31 11 6 21 27 34 23 21 21 7 31 34 20 24 34 15 30 31 27 7 32 20 24 3
#[49] 27 36 4 1 34 24 4 2 22 4 2 19 20 25 31 34 31 2 20 35 10 23 19 29
#[73] 32 33 32 23 37 2 32 24 31 2 19 2 1 24 10 26 37 9 11 37 2 17 27 30
#[97] 11 21 34 17
3 plot.ts(ts(number))
```

效果如图 5-2 所示。

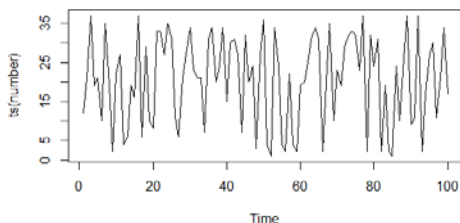


图 5-2

某地的彩票最后一个开奖号码按日期排好，就成了一个时间序列。在某些日子里总会有些人聚在街头巷尾去研究这个序列，恨不得头上能开了天眼，希望能看出晚上将会开什么号码，然后好赶紧下注。这很有研究精神，但统计学常识会告诉我们，已有的历史数据并不会影响下一期所抽中号码的概率，而这个概率只是服从均匀分布。也就是说，每个号码出现的概率，依然是平等的。一些私人庄家还会特别为最后一个号码（37 选 1）设置一个很诱人的赔率，譬如中 1 赔 33。有人要么眼红收赌注的庄家总是能吃香喝辣，要么怀疑幕后老板是不是“造码”（操纵结果）。其实，只要投注的人越多，投注码号越均匀，庄家的赔付比例就越接近 33/37，剩下的 4/37（大约 10%）的投注额就进了自己腰包，利润还是很丰厚的。换个角度说，投注得越多，亏得越惨。

说回数据挖掘研究，我们研究一个因变量随着自变量的变化规律，往往是假设因变量之间是不会相互影响的，而是相互独立的；但时间序列却恰恰相反，需要研究因变量之间的联系，譬如今天的股票收盘价格与昨天的收盘价格之间有很大的关联。

## 5.3 时间序列三件宝：趋势、周期、随机波动

时间序列分析主要找时间序列中是否存在这三样东西：趋势、周期、随机波动。

### 5.3.1 趋势

拿到一组数据，发现其中的趋势是最基本的要求。最简单的就是用平滑法（smoothing method），即计算时间序列的简单移动平均数，TTR 包的 SMA 函数可以实现，如下所示。

- SMA 函数

```
1 library(TTR)
2 plot(SS_close['2016/']) ###将 2016 年至今的上海证券交易所综合报收指数列出来
3 lines(SMA(SS_close['2015-12-01/'], n=30), col="green")
4 lines(SMA(SS_close['2015-12-15/'], n=10), col="blue")
5 lines(SMA(SS_close['2015-12-20/'], n=5), col="red")
###这三行是计算 2016 年来每天的指数移动平均数，其中绿色的计算周期是最近 30 个交易日，蓝
###色的是 10 天，红色的是 5 天
```

效果如图 5-3 所示。

这三根线是不是眼熟？对，它们就是 K 线图上的 30 日、10 日、5 日均线，凭借这些线我们可以大致了解到 2016 年的股指是一个触底反弹逐渐上升的过程。

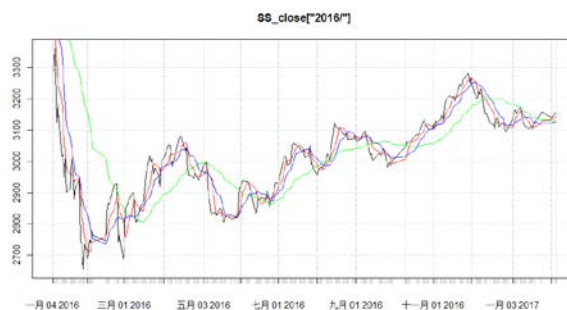


图 5-3

### 5.3.2 周期

一些地方每天的气温情况是典型的带有周期性的时间序列（如图 5-4 所示，数据来源：天气网 <http://www.tianqi.com/>）。譬如北京地区每日最高气温，呈现波形走势，基本上是以年作为周期，每年冬天处于波谷，每年夏天处于波峰，代码如下所示。

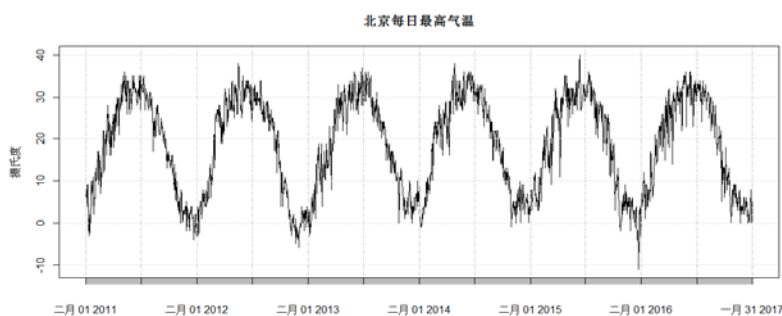


图 5-4

- 典型的季节分布

```
1 beijing_temperature = xts(scan("beijing.txt", sep = ","), order.by = seq.  
Date(as.Date("2011-02-01"), as.Date("2017-01-31"), "day")) ###这里在定义时间序  
###列时需要指定日期  
2 class(beijing_temperature)  
#[1] "xts" "zoo" ###时间序列可以保存为多种对象类型，我觉得保存为 xts zoo 可以比  
###较方便地引用不同时间段的数据。  
3 plot(beijing_temperature, ylab="摄氏度", main = "北京每日最高气温")
```



每年平均有 365 天，但 2012 年和 2016 年有 366 天，我们在构建序列的周期性时可以先把这两天数据去掉，对于整体的判断影响不大，代码如下所示。

- 时间序列分解

```
1 datelist = row.names(as.data.frame(beijing_temperature))
2 which(datelist == "2012-02-29")
#[1] 394
3 which(datelist == "2016-02-29")
#[1] 1855
4 bj_temp_ts = ts(beijing_temperature[c(-394,-1855)], start=c(2011,2,01),
frequency=365) ###调用分解函数前需要将对象转换成 ts 类
5 decom_bj_temp = decompose(bj_temp_ts)
6 > class(decom_bj_temp)
#[1] "decomposed.ts"
7 plot.ts(decom_bj_temp)
```

效果如图 5-5 所示。

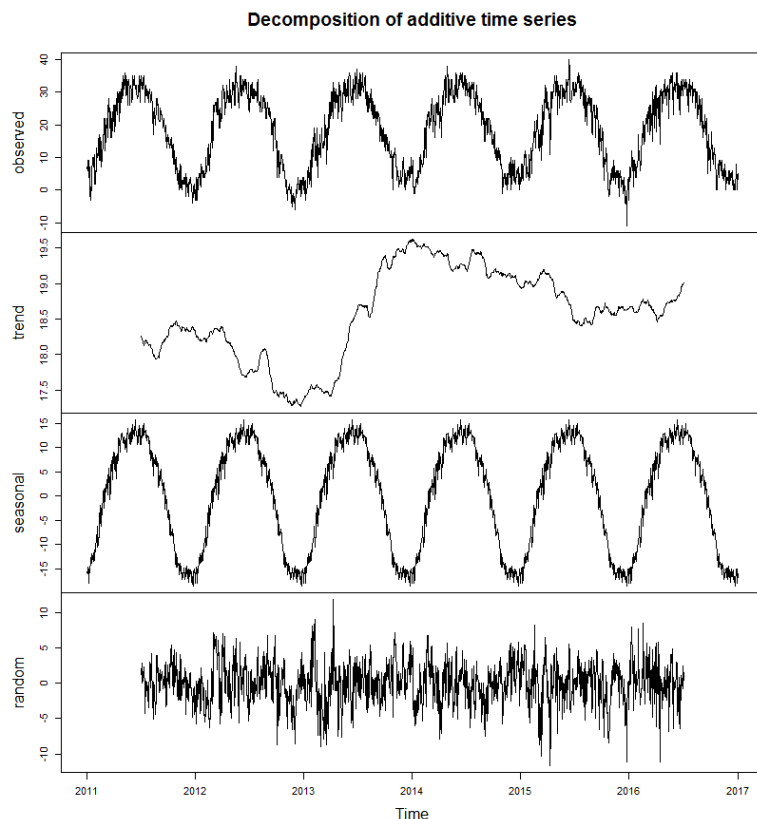


图 5-5

这样一来，我们就可以将数据分解成三个部分了：trend（趋势）、seasonal（周期性）和 random（随机波动）。将这三部分加起来，就是最上边的 observed（观测值）。

### 5.3.3 随机波动

5.3.2 节我们提到了时间序列的第三个部分是随机波动（random）。一个时间序列可以不存在周期性，也可以不存在趋势，但必定存在着不规则波动的部分。这部分数据已经将时间序列中与时间相关的部分去掉，理论上说数据之间不再存在自相关的联系，而应该符合正态分布，代码如下所示。

- 正态分布检验

```
1 r=decom_bj_temp$random
2 hist(r, breaks= 50, probability = T, main="") ###画出随机波动项的分布
3 curve(dnorm(x, mean(r), sd(r)), from = -12, to = 12, add=T, col="red")
###画出同样大小均值和标准差的正态分布曲线，两者吻合得比较好
```

效果如图 5-6 所示。

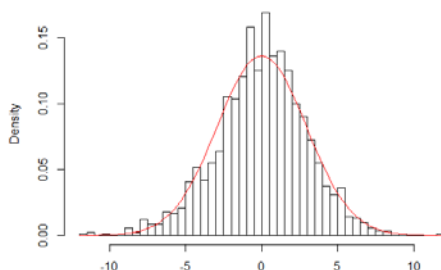


图 5-6

## 5.4 预测分析

以史为鉴，可见未来，时间序列分析除了可以让我们一眼就看到事情的发生历程外，还有一个重要的功能就是预测事情未来的发展趋势。用于时间序列的预测方法有很多种，这里介绍两种比较简单的方法：指数平滑法和 ARIMA 模型。

### 5.4.1 指数平滑法

为了理解序列分析中的一些基本的原理，我们先给大家介绍简单指数平滑法。5.3.1 节中我们分析序列的趋势时用了平滑法来计算股票的 5 日、10 日、30 日平均价格，这里我们可以基于这种思路对未来的数据进行简单的预测。该模型的原理很简单：

$$F_{t+1} = \alpha Y_t + (1 - \alpha)F_t \quad (1)$$

其中 $F_{t+1}$ 是时间点  $t+1$  的预测值,  $Y_t$ 是时间点  $t$  的观测值,  $\alpha$ 是权重系数, 取值范围是从 0 到 1。 $F_t$ 也可以写成

$$F_t = \alpha Y_{t-1} + (1 - \alpha)F_{t-1} \quad (2)$$

把②式代入①式中, 可以表示为

$$F_{t+1} = \alpha Y_t + (1 - \alpha)\alpha Y_{t-1} + (1 - \alpha)^2 F_{t-1} \quad (3)$$

迭代下去则有:

$$F_{t+1} = \alpha Y_t + (1 - \alpha)\alpha Y_{t-1} + (1 - \alpha)^2 \alpha Y_{t-2} + \cdots + (1 - \alpha)^n \alpha Y_{t-n} + \cdots + (1 - \alpha)^t \alpha F_1 \quad (4)$$

我们可以看到④式中出现序列中所有的观测值, 说明预测值 $F_{t+1}$ 受到序列中每个观测值的影响。但由于 $(1 - \alpha)$ 的取值范围是小于 1 的正数, 则 $(1 - \alpha)^n$ 项会随着  $n$  的增大而越来越小, 时间  $n$  之前的点对于预测值 $F_{t+1}$ 的影响越来越小。

在 R 里边实现指数平滑法的操作也很简单, 我们以 2016 年第 4 季度的上证综合指数为例。先利用①式构建一个指数平滑法的函数, 代码如下所示。

- 指数平滑法

```
1 SSQ4=SS_close['2016-10/2016-12']
2 plot(as.vector(SSQ4), col="blue", type="l",xlab="Day",ylab="收盘指数")
3 points(as.vector(SSQ4), col="blue")
4 pred[1]=mean(SS_close['2016-09-28/2016-09-30'])###第一个预测值为前三个交易
                                     ###日的平均值

5 a=0.9
6 for(i in 2:length(SSQ4)){
7   pred[i]<-a*SSQ4[i-1]+(1-a)*pred[i-1]
8 }
9 points(pred, col="green", type="l")
10 points(pred, col="green")
11 a=0.6
12 for(i in 2:length(SSQ4)){
13   pred[i]<-a*SSQ4[i-1]+(1-a)*pred[i-1]
14 }
15 points(pred, col="red", type="l")
16 points(pred, col="red")
```

效果如图 5-7 所示。

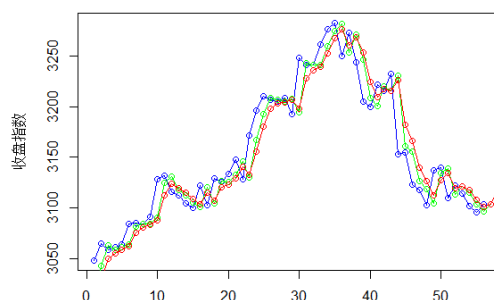


图 5-7

这三根线是每天的沪市收盘指数、权重  $\alpha=0.9$  时的预测走势。权重  $\alpha=0.6$  时的预测走势。可以看出三根线的走势基本上是一致的，但我们也该注意到，同一天的预测值的走势其实是有一定的滞后性的。用指数平滑法来预测大市的走势显然是过于简单的，当观察值出现拐点时，预测的质量会下降。这里举这个方法作为例子，只是展示一下时间序列预测模型的建立模式。基于这个方法还发展和优化出很多模型，包括 Holt 双参数线性指数平滑法、Winter 线性和季节性指数平滑法等。

## 5.4.2 ARIMA 模型预测

ARIMA 模型是一个比较常见的预测模型，我们也可以用这个模型来对沪市收盘指数进行预测。开始之前，我们需要获得一个平稳的时间序列，可以对其进行差分处理，如下所示。

- 季节拆分图

```
1 plot(diff(SSQ4,differences=1))
2 plot(diff(SSQ4,differences=2))
```

效果如图 5-8 所示。



图 5-8

图 5-8 左图是 1 阶差分，右图是 2 阶差分，看起来右图比左图更平稳一些，因此，我们对时间序列进行两次差分以得到平稳序列。

接着，就是要找到适合的 ARIMA 模型了，即寻找  $ARIMA(p,d,q)$  中适合的  $p$  值和  $q$  值 ( $d$  值是

指  $n$  阶差分, 上一步通过 2 阶差分得到稳定序列, 所以  $d=2$  ), 我们可以通过检查时间序列的自相关图和偏相关图得到这两个值, 代码如下所示。

• 时间序列 acf 分析

```
SSQ4diff2=diff(SSQ4,2)
acf(SSQ4diff2,lag.max=30,na.action=na.pass)
acf(SSQ4diff2,lag.max=30,na.action=na.pass,plot=FALSE)
#Autocorrelations of series 'SSQ4diff2', by lag
#
#      0      1      2      3      4      5      6      7      8      9     10
# 1.000 0.471 0.028 0.079 0.060 0.042 0.142 0.305 0.215 0.085 0.021
#     11     12     13     14     15     16     17     18     19     20     21
#-0.045 -0.008 -0.060 -0.098 -0.048 -0.056 -0.026  0.016  0.008 -0.174 -0.262
#     22     23     24     25     26     27     28     29     30
#-0.117 -0.068  0.012  0.056 -0.008 -0.036 -0.121 -0.158 -0.082
```

效果如图 5-9 所示。

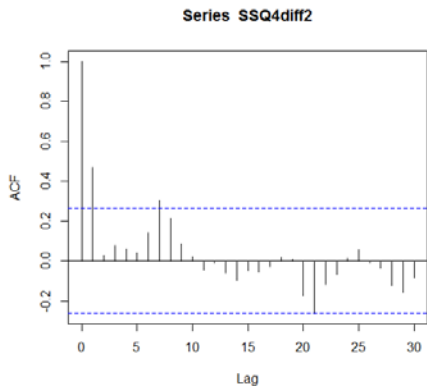


图 5-9

自相关图显示滞后 1 阶和 7 阶自相关值都超出边界, 1 阶要比 7 阶显著, 其他值都没有超出显著边界, 代码如下所示。

• 时间序列 pacf 分析

```
pacf(SSQ4diff2,lag.max=30,na.action=na.pass)
pacf(SSQ4diff2,lag.max=30,na.action=na.pass,plot=FALSE)
#
#Partial autocorrelations of series 'SSQ4diff2', by lag
#
#      1      2      3      4      5      6      7      8      9     10     11
# 0.471 -0.249  0.246 -0.140  0.139  0.078  0.277 -0.104  0.130 -0.167  0.040
#     12     13     14     15     16     17     18     19     20     21     22
#-0.044 -0.150 -0.078 -0.050 -0.070  0.088  0.021  0.027 -0.216  0.024 -0.016
#     23     24     25     26     27     28     29     30
```

```
# 0.028 0.092 -0.052 0.008 0.059 -0.090 -0.030 0.001
```

效果如图 5-10 所示。

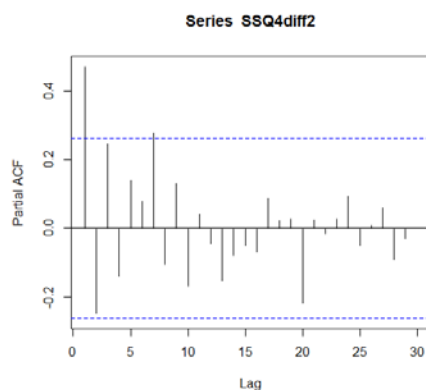


图 5-10

偏自相关值选 1 阶，故我们的 ARIMA 模型为 `arima(1, 2, 1)`，代码如下所示。

- 构建 arima 模型

```
1 SSQ4arima=arima(SSQ4, order=c(1,2,1))
2 SSQ4arima
#
#Call:
#arima(x = SSQ4, order = c(1, 2, 1))
#
#Coefficients:
#      ar1      ma1
#    -0.2240 -0.8995
#s.e.   0.1414  0.0740
#
#sigma^2 estimated as 481.1: log likelihood = -244.42, aic = 494.83
```

预测一下 5 个交易日沪市收市指数，并和实际值比较，代码如下所示。

```
1 forecast.Arima(SSQ4arima, h=5)
#   Point Forecast   Lo 80   Hi 80   Lo 95   Hi 95
#57      3096.437 3068.327 3124.548 3053.446 3139.429
#58      3092.537 3055.158 3129.916 3035.371 3149.704
#59      3087.897 3041.051 3134.744 3016.251 3159.544
#60      3083.423 3027.655 3139.191 2998.134 3168.713
#61      3078.912 3014.300 3143.524 2980.097 3177.727
2 observation=SS_close['2017'][1:5]
3 plot.forecast(SSQ4arimaforecast)
4 points(x=seq(57,61,1),y=observation, col="red")
```

效果如图 5-11 所示。

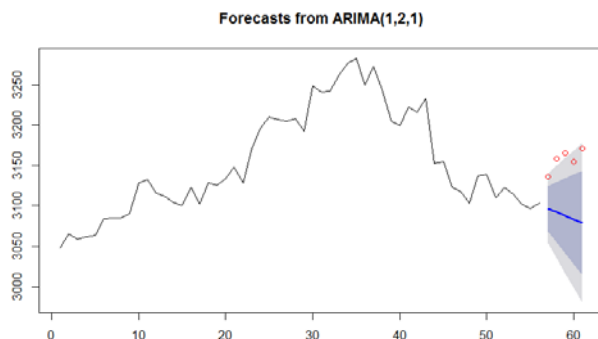


图 5-11

从图 5-11 中可以看出元旦后趋势上涨，实际观察到的收盘指数在预测值的上方 95% 区间了。对于 ARIMA 的 3 个参数我们这里提出一个比较暴力的方法，一般 3 个参数都是整数，而且其范围有限，一般在 0~12 之间（当然也有长周期，可以根据业务问题大概估计），这样 3 个参数的组合就是有限的，我们拿所有可能的组合建模然后测试，就能获得最优的模型参数，从而忽略上述选取参数的复杂过程。

时间序列分析的预测方法和模型有很多，除了涉及数理统计模型外，还要深入地理解研究对象的客观运行规律，这样才能提高模型预测的准确率。这里只是简单地列举了一些入门的知识，想深入学习最好参考更多的资料，但就我经验而言，时间序列的模型实在是个鸡肋，如果它们能做得好，那些更加复杂的模型可能做得更好，主要问题还是在于你怎么创造出包含各类信息的变量。

## 第 6 章

# 选择什么算法也有一套流程

### 6.1 重新审视一下这几个模型

在数据建模过程中，我们能够使用的算法和模型并不是太多，而 Logistic 回归、随机森林、神经网络则是经常出现在工具箱里的模型，但我把这些称为比较高级的模型，其实真正用到它们的时候并不多，但是一旦用到就无法逃避模型分析的过程。

#### 6.1.1 Logistic 回归

总的来说，逻辑回归分析是对定性变量的回归分析。

在社会科学中，最流行的模型非 Logistic 回归莫属。Logistic 回归分析根据因变量取值类别不同，又可以分为 Binary Logistic 回归分析和 Multinomial Logistic 回归分析。

在现实生活中，当某一个预测变量在无穷小到接近 0 值的区间范围时，其对应的期望值很小，并且预测变量从无穷小变化到 0 的时候，它所对应的期望值变化较慢，而在无穷大区间范围时，期望值无限接近于 1，它的期望值变化同样较慢，唯有在 0 区间范围时，我们的期望值变化较大、较明显。在数学上看来，我们需要一个函数  $f(p)$  对  $x$  的变化在  $f(p)=0$  或者  $f(p)=1$  的附近是不敏感、缓慢的，且非线性的程度较高。针对这种情况，逻辑回归引入了 Logit 变换。比如在信用评分领域最常用的模型：评分卡模型，其实评分卡模型的原型就是逻辑回归，只是在处理变量时比较特殊。

不同于决策树或者随机森林等算法在模型训练过程中自动筛选对结果存在显著的变量，逻辑回归并不会自动去掉无意义的变量。当然，学过回归的读者应该想到，可以用前向变量逐步回归或者后向逐步回归等变量筛选方法在逻辑回归中进行变量筛选，但我们并不建议一开始就使用逐步回归。因为逐步回归只是选择 BIC 最大的变量组合进入模型，所选取变量对业务问题的解读意义并不强，并且逐步回归算法所选取的变量集并不是最优集合，仅仅是一个启发性的算法。而评分卡算法计算一个叫作信息价值（IV 值）的指标作为变量筛选的依据。在介绍 IV 值之前，我们先来看看评分卡



模型是怎么进行变量处理的。

在计算 IV 值之前是需要分箱的,所谓分箱就是对连续变量的离散化,所谓离散化就是把连续变量切割为分类变量(有些武断),也就是分组。为什么要进行分箱呢?分箱主要是为了解决分类模型的非线性问题。逻辑回归的公式中的主体部分实质上是一个线性函数,包裹这个线性函数的是一个单增的函数。因此,当预测变量与目标变量不是线性关系时,模型的表现并不一定会很好。

例如,预测变量  $x$  本身对目标变量  $y$  不显著,但经过二次变换后对目标变量  $y$  显著的时候,我们常常会在回归方程里面加入变量  $x$  的二次项。然而,这需要对变量  $x$  测试各种变换后才能得到对变量  $y$  显著的衍生变量,例如  $x$  的对数变换、三次方变换等,这往往非常浪费分析时间,而通过对变量的分箱,只要分箱粒度足够小,我们能使变量适应任何曲线。

为了更清晰地说明问题,下面做如下几点假设。

- (1) 假设现在的分类问题里只有一维连续特征(即  $x$  为单维向量)。
- (2) 假设真实决策面为简单二次函数,即

$$y < -5x^2 + 500x + 100$$

那么,真实的决策面应该是如图 6-1 所示的分解曲线。

很显然,如果就拿  $x$  进行训练,训练出的模型只能是一维线性模型,这样的模型效果肯定是不好的,因为它可能会是这个样子(如图 6-2 所示直线)。

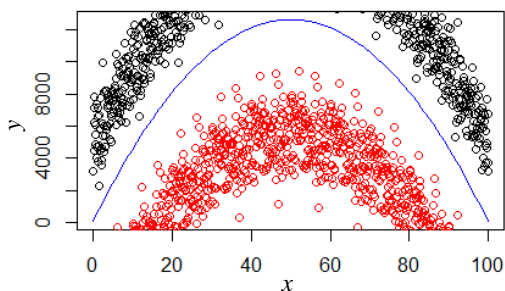


图 6-1

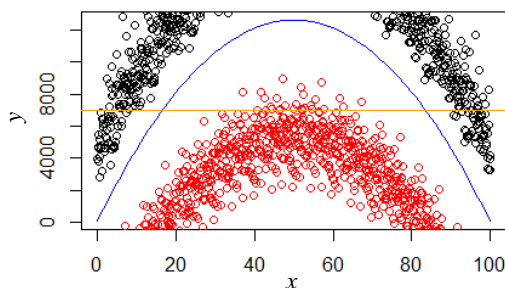


图 6-2

但是,如果我们将  $x$  进行分段处理,将  $x$  分箱,离散成 0/1 向量,利用新的特征向量 ( $x_1, x_2, x_3, \dots, x_n$ ) 训练得到模型将会有如下形式:

$$y = \begin{cases} y_1, x < x_1 \\ y_2, x < x_2 \\ y_3, x < x_3 \\ \dots \\ y_n, x < x_n \end{cases}$$

新的模型在原来的坐标轴下会长成这个样子(水平短线为分箱段),如图 6-3 所示。

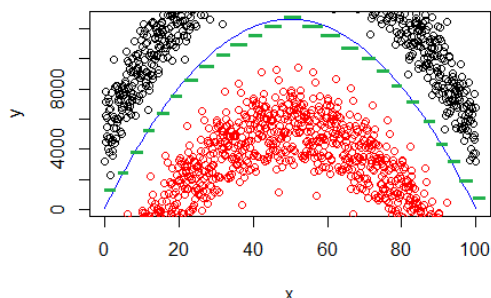


图 6-3

如果将  $x$  做最大程度的细分分箱，理论上对任何曲线都可以拟合得很好！因此将特征离散成高维的布尔特征可以解决逻辑回归中的非线性问题，以上就是要进行分箱处理的原因所在。评分卡模型在分箱的基础上更进一步。在分箱过后，计算证据权重 WOE 的数值来替换分箱后出现的高维变量空间，用于衡量目标变量归属不同分类时的概率差异，而且分箱后会有一些表现相近的箱被合并，从而平滑变量的变化趋势。证据权重 WOE 的计算公式如下：

$$WOE_i = \ln\left(\left(\frac{g_i}{g}\right) / \left(\frac{b_i}{b}\right)\right)$$

其中，

$WOE_i$  表示变量第  $i$  个分箱对应的 WOE 值；

$g_i$  表示变量第  $i$  个分箱对应的好客户数；

$b_i$  表示变量第  $i$  个分箱对应的坏客户数；

$g$  表示样本中总的好客户数；

$b$  表示样本中总的坏客户数。

WOE 值越大，代表该分组中对变量  $y=1$  表现更显著。

有了 WOE，接下来我们回到 IV 值的计算上。

IV 的计算公式为：

$$IV = \sum_i \left( \frac{g_i}{g} - \frac{b_i}{b} \right) \times WOE_i$$

变量的 IV 值越大，表示该变量在目标变量  $y$  上的分布差异越大，变量的区分能力越强，即预测能力无敌。

一般情况下，我们选取 IV 值大于 0.02 的变量进入模型当中。但当一个变量的 IV 值达到 0.5 的时候，我们一般不会选这个变量进入模型，而是将这个变量作为分群变量，从而将样本分为几个群体，分别对不同的群体开发评分卡模型，并且分出的各个群体要有业务上的解读意义，即站在业务的角度可以解释。

需要注意的是，评分卡模型一开始是在信用卡行业应用的，但是，评分卡模型是具有通用性的，从上文我们可以知道，评分卡的核心在于逻辑回归和 WOE 转换，在我实际的应用过程中，逻辑回归的性能就比很多模型要稳定优越，而 WOE 转化是一个非常巧妙的变换，既解决了逻辑回归的局限性，又不用单纯的分箱那样造成高维空间求解。评分卡主要可以分为以下 4 类。

(1) 审批评分，这类评分卡就是评分卡创建一开始的业务场景。根据客户的基本信息（性别、年龄、工作、学历等）和少量的金融信息（总资产、总负债等）来对客户进行评分。一般情况下，评分越高，客户越好，信用卡违约的可能性越低。

(2) 违约预警，主要对客户在未来一段时间内逾期违约的可能性进行预测，评分越高，逾期违约的可能性越高。

(3) 收益评分，收益评分主要是在客户成为企业的用户后进行预测，从而判断客户未来一段时间对公司的投入。例如信用卡领域，信用卡中心会允许一部分客户产生少量的逾期，但这部分客户又不会成为呆账客户（这类人才是信用卡营收的重要贡献者），而在其他领域一般用于判断客户的价值度，提前预测客户是否会追加投资等应用场景。通常情况下，评分越高，公司收益越大，客户越有价值。

(4) 流失评分，主要是针对客户在未来一段时间的流失进行预测，帮助业务人员制定相关的挽留政策。如果将流失的可能性作为目标变量，则评分越高，流失可能性越大。

## 6.1.2 我要的不是一棵树，而是整座森林：随机森林

随机森林是指利用多棵树对样本进行训练、预测的一种分类器，由于它需要调整的参数较少、分类速度很快，不容易过度拟合（不等于不会出现过度拟合的情况），能高效处理大样本数据，而且能估计哪个特征在分类中更重要，较强的抗噪音能力等特点，因此在建模工作中受到广泛好评。

简单来说，随机森林就是由多棵 CART（Classification And Regression Tree）构成的。对于每棵树，它们使用的训练集是从总的训练集中又放回再重新采样出来的，这表示总的训练集中的有些样本可能多次出现在一棵树的训练集中，也可能从未出现在一棵树的训练集中，而在训练每棵树的节点时，使用的特征将从所有特征中按照一定比例随机地无放回抽取。随机森林通过一种自助法重采样技术生成很多个树分类器，其步骤如下：

(1) 从原始训练数据中生成  $k$  个自助样本集，每个自助样本集是每棵分类树的全部训练数据。

(2) 每个自助样本集生长为单棵分类树。在树的每个节点处从  $M$  个特征中随机挑选  $m$  个特征，按照节点不纯度最小的原则从  $m$  个特征中选出一个特征进行分支生长。这棵分类树进行充分生长，使每个节点的不纯度达到最小，不进行通常的剪枝操作。

根据生成的多个树分类器对新的数据进行预测，分类结果按每个树分类器的投票多少而定，随机森林有两个重要参数：一个是树节点预选的变量个数；另一个是随机森林中树的个数。一般情况下，树的节点个数默认为总变量个数的  $1/2$  次方。我们建议模型的最终变量不宜过多，最好控制在 10~20 个，太多的变量不利于后续的解释。所以在刚开始进行变量筛选和输入时，需要调节这个参数，而随机森林树数则要根据训练集和测试集的结果来调整。我们先把数据分为训练集和测

试集，随机给这个参数赋一个值，例如 100，等模型训练出来后，对测试集进行预测，计算测试集的准确率，当测试集的准确率明显低于训练集时（这里的“明显”是需要交叉检验的），我们就要把这个参数调低一点，因为出现了过拟合。反复进行这个过程，直到测试集的准确率和训练集相当。

除了上述两个调优参数需要重点关注外，我们还需要对输入变量进行检验。这不同于评分卡模型，在训练逻辑回归之前就进行了变量筛选，随机森林在训练结束之后才计算每个变量的重要性。因此，我们在进行第一轮随机森林训练时要把所有变量都纳入模型输入。

随机森林中我们使用两个变量的重要性指标：

（1）均方误差的增加（increase in MSE），计算时对每一个变量，比如  $X_1$  随机赋值，如果  $X_1$  重要，预测的误差会增大。所以这个指标越大，变量越重要。需要注意的是，有些文献会有一个叫作平均准确率降低度（Mean Decrease Accuracy）的指标。这两个指标是等同的，因为均方误差的增加就等同于准确性的减少。

（2）节点纯度的降低（IncNodePurity）。如果是分类，节点纯度的降低就是 Gini 系数减少。如果是回归，节点纯度降低其实就是 RSS 的减少。节点纯度越高，变量越重要。同样，有些文献会有一个叫 Mean Decrease Gini 的指标，其实节点纯度增加就等同于 Gini 指数的减少，也就是节点里的数据或 class 都一样，这两个指标也是等同的。

在一轮训练过后，我们就可以根据这两个指标来进行初步的变量筛选，例如分别取这两个指标最显著的 20 个变量，然后取它们的并集来进行我们第 2 轮的随机森林训练。

### 6.1.3 神奇的神经网络

我们的大脑可视作为 1000 多亿神经元组成的神经网络，神经元的信息传递和处理是一种电化学反应。树突由于电化学反应接受外界的刺激；通过胞体内的活动体现为轴突电位，当轴突电位达到一定的值时，则形成神经脉冲或动作电位；再通过轴突末梢传递给其他的神经元（见图 6-4）；从控制论的观点来看，这一过程可以看作一个多输入单输出非线性系统的动态过程，并且有 3 大特点：

① 巨量并行性；② 信息处理和存储单元结合在一起；③ 自组织自学习功能。

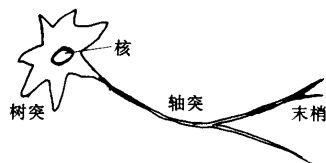


图 6-4

由此，人们根据人体神经元、神经网络结构，设计出人工神经网络模型，由具有适应性的简单单元组成的广泛并行互联的网络，它的组织能够模拟生物神经系统对真实世界的物体所做出的交互反映。

人工神经网络以其具有自学习、自组织、较好的容错性和优良的非线性逼近能力，受到众多领

域学者的关注。在实际应用中，80% ~ 90% 的人工神经网络模型是采用误差反传算法或其变化形式的网络模型，目前主要应用于函数逼近、模式识别、分类和数据压缩或数据挖掘方面。

- 优点：
- (1) 较强的容错性；
  - (2) 很强的自适应学习能力；
  - (3) 可将识别和若干预处理融为一体进行。

## 6.2 银行信用卡评估模型之变量筛选

这次，我们使用某银行信用卡中心的客户数据来进行研究，银行拥有客户非常详细的信息，包括客户的基本信息（年龄、性别、职业等）。这些信息不仅在信用卡领域可以使用，还可以在银行的其他产品线上使用。例如平安银行除了拥有信用卡业务外，还具有储蓄业务、保险业务、证券业务等。当一个客户购买了其中一个产品线上的业务后，其他产品业务在笼络客户时，可以首先考虑这个已有的客户，因为我们已经拥有了这个客户非常详尽的信息。

### 6.2.1 变量构建

假设这家银行现在想推销它的银行卡，它要求信用卡中心提供一批客户信息作为目标客户范围，显然，这属于典型的交叉营销业务场景。

在建模前首先要构造变量，一般来说，构造对目标变量有影响的预测变量需要建模人员具有一定的业务知识。我建议在变量构造过程中，可以和项目组的人进行头脑风暴，最好邀请业务人员参与，先构造一版变量列表，然后对这些变量进行探索。本次一共选择了 10 个变量。如表 6-1 所示。

表 6-1

变 量	备 注
年龄	客户年龄
工作	工作类型（分类：“管理”“未知”“失业”等）
婚姻	婚姻状况（分类：“已婚”“离婚”“单身”。注：“离婚”是指离婚或丧偶）
教育	（分类：“未知”“中学”“初级”“大专”）
默认	有无信用违约（二进制：“是”“否”）
余额	平均年度余额：欧元（数字）
住房贷款	有房屋贷款吗（二进制：“是”“否”）
贷款	有个人贷款（二进制：“是”，“否”）
以前活动响应度	以前营销活动的结果（分类：“未知”“其他”“失败”“成功”）
客户是否订购定期存款	目标变量

我们先使用 Logistic 回归完成对用户是否接受定期存款业务进行预测。

### 6.2.2 Logistic 回归变量筛选

变量筛选与其说是一门科学，不如说是一门艺术（表示遇到这个问题就无从下手了），但基本上遵循几个原则：变量的简单性、变量对结果的可解释性（被业务人员挑战时可以说服他们）。但作为比较理性的人，我们更加关注变量的预测能力，而 IV 值刚好用于衡量变量的预测能力。我们先对连续型变量进行分箱，分别计算各箱的 WOE 值，然后计算 IV 值，如下所示。

- 连续变量分箱

```
1 mydata<-read.csv("H:\\\\探寻数据背后的逻辑 R 语言数据挖掘之道\\第 6 章选择什么算法也有  
有一套流程\\data\\bank1.csv", header=T)  
2 mydata <- mydata[mydata$age>20,] ##过滤掉小于 20 岁的用户，量较少  
3 mydata$age <- mydata$age%/%10  
4 mydata$balance <- mydata$balance%/%1000  
5 mydata$age <- as.factor(mydata$age)  
6 mydata$balance <- as.factor(mydata$balance)
```

由于变量列表中只有年龄和年度余额是连续型变量，所以我们只需要对这两个变量进行分箱。我们以 10 为单位对年龄进行分箱，以 1000 为单位对年度余额进行分箱。注意，R 中的“%/%”符号是整除的意思。还需要将年龄和年度余额转为因子型变量，因为我们后续计算 WOE 的函数要求预测变量均为因子变量。

- 计算 WOE 值

```
1 library(klaR)  
2 woemodel <- woe(y~., data = mydata, zeroadj=0.5, applyontrain = TRUE)  
3 woemodel$woe  
4 agewoe <- data.frame(woemodel$woe$age)  
5 agewoe  
# woemodel.woe.age  
# 2      -0.3043389  
# 3       0.1286018  
# 4       0.1294581  
# 5       0.1724998  
# 6      -1.0703419  
# 7      -1.5661498  
# 8      -1.7553918  
6 IV <-data.frame(woemodel$IV)
```

上述代码中，使用 klaR 包里面的 WOE 函数计算 WOE 值和 IV 值，结果如图 6-5 所示（由于 WOE 值过多，我们只输出年龄段的 WOE 值）。

从图 6-5 中可以看出年龄在 50~60 岁的客户最有可能购买我们的储蓄产品。这部分客户面临退休养老的问题也确实最有可能购买储蓄产品，与常识相一致。我们看一下不同变量的 IV 值，如图

6-6 所示。

箱	WOE
2	-0.3043389
3	0.1286018
4	0.1294581
5	0.1724998
6	-1.0703419
7	-1.5661498
8	-1.7553918

图 6-5

变量	IV 值
age	1.29E-01
job	1.27E-01
marital	3.83E-02
education	3.34E-02
default	2.20E-05
balance	1.01E-01
housing	1.04E-01
loan	6.01E-02
poutcome	4.67E+00

图 6-6

从图 6-6 中可以看出是否对历史的营销活动做出过响应这个变量最显著，其次为年龄 (age)，最不显著的变量为是否有违约记录。一般情况下，当变量小于 0.02 阈值时，我们剔除这个变量。当然，当大于 0.02 阈值的变量过多时，我们也可以适当地把阈值提高。在这个数据集下，我们仍然用 0.02 作为阈值，那么就只有是否有违约记录这个变量被剔除。

由于 WOE 函数已经帮我们进行了 WOE 转化，所以只需要将转化后的预测变量值和目标变量重新包装成一个 data.frame 就可以了，如下所示。

• WOE 转化

```
1 mydata.b <- data.frame(woemodel$xnew, mydata$y)
```

在模型训练之前，我们需要将数据集分为训练集和测试集。我们将 70% 的数据划分给训练集，剩下的 30% 划分为测试集。之后，我们调用 glm 函数进行逻辑回归建模，注意连接函数是 Logit 变换，如下所示。

• 逻辑回归建模

```
1 mydata.b <- data.frame(woemodel$xnew,y=mydata$y)
2 set.seed(123)
3 nr <- sample(nrow(mydata.b), 0.7*nrow(mydata.b), replace = FALSE, prob =
NULL)
4 train <- mydata.b[nr,]
5 test <- mydata.b[-nr,]
6 lg <- glm(y~.,family = binomial(link = logit),data = train)
```

通过 summary 逻辑回归得到的结果如下所示。

• 查看模型的基本情况

```
1 summary(lg)
# Call:
```

```
# glm(formula = y ~ ., family = binomial(link = logit), data = train)
#
# Deviance Residuals:
#      Min       1Q   Median       3Q      Max
# -2.3049  -0.4714  -0.3836  -0.3073   2.7130
#
# Coefficients:
#              Estimate Std. Error z value Pr(>|z|)
# (Intercept)   -2.06204    0.06166  -33.445 < 2e-16 ***
# woe.age       -0.92173    0.17253   -5.343 9.17e-08 ***
# woe.job       -0.15496    0.19894   -0.779 0.436025
# woe.marital   -1.14500    0.30783   -3.720 0.000200 ***
# woe.education -0.86532    0.34521   -2.507 0.012188 *
# woe.default   -12.43726   12.34306   -1.008 0.313631
# woe.balance   -0.55351    0.19160   -2.889 0.003867 **
# woe.housing   -0.64319    0.19436   -3.309 0.000935 ***
# woe.loan      -0.68040    0.27235   -2.498 0.012480 *
# woe.poutcome  -0.99214    0.08239  -12.042 < 2e-16 ***
# ---
# Signif. codes:
# 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 1)
#
#      Null deviance: 2232.9 on 3158 degrees of freedom
# Residual deviance: 1949.7 on 3149 degrees of freedom
# AIC: 1969.7
#
# Number of Fisher Scoring iterations: 5
```

不要忘记，除了是否曾经有违约记录和工作类型这两个变量不显著，其他各个变量都显著。我们使用逐步回归来剔除部分不显著变量，代码如下所示。

- 逐步回归剔除变量

```
1 lg2 <- step(lg)
# Start: AIC=1969.66
# y ~ woe.age + woe.job + woe.marital + woe.education + woe.default +
#      woe.balance + woe.housing + woe.loan + woe.poutcome
#
#              Df Deviance    AIC
# - woe.job      1  1950.3 1968.3
# - woe.default  1  1950.6 1968.6
# <none>          1949.7 1969.7
# - woe.education 1  1955.9 1973.9
# - woe.loan      1  1956.6 1974.6
```



```

# - woe.balance      1    1957.8 1975.8
# - woe.housing      1    1960.6 1978.6
# - woe.marital      1    1963.4 1981.4
# - woe.age          1    1976.9 1994.9
# - woe.poutcome     1    2099.2 2117.2
#
# Step: AIC=1968.26
# y ~ woe.age + woe.marital + woe.education + woe.default + woe.balance +
#     woe.housing + woe.loan + woe.poutcome
#
#               Df Deviance   AIC
# - woe.default   1    1951.2 1967.2
# <none>          1950.3 1968.3
# - woe.loan      1    1957.3 1973.3
# - woe.education 1    1958.2 1974.2
# - woe.balance   1    1958.6 1974.6
# - woe.housing   1    1962.7 1978.7
# - woe.marital   1    1964.1 1980.1
# - woe.age       1    1987.5 2003.5
# - woe.poutcome  1    2100.3 2116.3
#
# Step: AIC=1967.16
# y ~ woe.age + woe.marital + woe.education + woe.balance + woe.housing +
#     woe.loan + woe.poutcome
#
#               Df Deviance   AIC
# <none>          1951.2 1967.2
# - woe.loan      1    1958.0 1972.0
# - woe.education 1    1959.0 1973.0
# - woe.balance   1    1959.1 1973.1
# - woe.housing   1    1963.7 1977.7
# - woe.marital   1    1965.3 1979.3
# - woe.age       1    1988.1 2002.1
# - woe.poutcome  1    2100.6 2114.6
2 summary(lg2)
# Call:
# glm(formula = y ~ woe.age + woe.marital + woe.education + woe.balance +
#     woe.housing + woe.loan + woe.poutcome, family = binomial(link = logit),
#     data = train)
#
# Deviance Residuals:
#      Min       1Q   Median       3Q      Max
# -2.2911  -0.4753  -0.3861  -0.3143   2.6887
#

```

```
# Coefficients:
#           Estimate Std. Error z value Pr(>|z|)
# (Intercept)  -2.06334    0.06162 -33.484 < 2e-16 ***
# woe.age      -0.97852    0.15357  -6.372 1.87e-10 ***
# woe.marital  -1.15967    0.30758  -3.770 0.000163 ***
# woe.education -0.93670    0.33195  -2.822 0.004775 **
# woe.balance  -0.54460    0.19098  -2.852 0.004349 **
# woe.housing  -0.67285    0.19061  -3.530 0.000416 ***
# woe.loan     -0.67148    0.27176  -2.471 0.013478 *
# woe.poutcome -0.99037    0.08228 -12.037 < 2e-16 ***
# ---
# Signif. codes:
# 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 1)
#
# Null deviance: 2232.9 on 3158 degrees of freedom
# Residual deviance: 1951.2 on 3151 degrees of freedom
# AIC: 1967.2
#
# Number of Fisher Scoring iterations: 5
```

AIC 信息准则即 Akaike Information Criterion，是衡量统计模型拟合优良性的一种标准。它建立在熵的概念基础上，可以权衡所估计模型的复杂度和此模型拟合数据的优良性。AIC 越小，模型越优越。从上述结果可以发现，AIC 从一开始的 1969.6 降低为 1967.2，所有的预测变量都变为显著。在所有变量中，是否结婚这个变量对目标变量最显著。或许你会说，这与 IV 值最大为年龄这个变量不一致了。为什么会不一致呢？其实逻辑回归考虑所有变量的重要性，是在全局上的一个把握，而 IV 值的计算仅仅考虑单个变量，并没有考虑到偏相关的问题。

## • 预测测试集

```
p_log_test <- predict(lg2,test)
p_log_test <- exp(p_log_test)/(1+exp(p_log_test))
table(cut(p_log_test,breaks = seq(0,1,0.01)))
p_log_test_01 <- c()
p_log_test_01[p_log_test < 0.1] = "0"
p_log_test_01[p_log_test >= 0.1] = "1"
table(p_log_test_01 ,test$y)
# p_log_test_01 no yes
#           0 813  69
#           1 382  91
```

predict 输出的结果并不是 Logistics 回归最后的计算结果，我们需要将 predict 的结果代入 logit 函数求出最后的概率值。注意，0.1 这个阈值是我们测试出来的，综合考虑查全率和准确率权衡得出的。尽管模型的准确率很一般，但衡量模型性能的指标不仅仅有准确率，这个我们后续会讲。

### 6.2.3 随机森林变量筛选

我们再次尝试使用随机森林完成相同的建模任务，并提取变量的重要性，如下所示。

- 随机森林变量的重要性

```
1 library(randomForest)
2 fit.rf <- randomForest(y~.,data = train, ntree = 100, mtry = 9, importance
= TRUE)
3 importance <- fit.rf$importance
4 importance
#
```

	no	yes
# woe.age	0.0094247225	0.0013360532
# woe.job	0.0079701320	-0.0269833290
# woe.marital	0.0059072449	-0.0001815758
# woe.education	0.0026593799	-0.0100938039
# woe.default	-0.0003404273	-0.0019415286
# woe.balance	0.0015085625	0.0017399314
# woe.housing	0.0026178674	0.0005469139
# woe.loan	-0.0007440781	0.0023200606
# woe.poutcome	0.0121450503	0.0708647889

```
#
# MeanDecreaseAccuracy MeanDecreaseGini
# woe.age 0.0085087986 71.006289
# woe.job 0.0040973703 84.599147
# woe.marital 0.0052429778 36.791440
# woe.education 0.0012623049 50.940173
# woe.default -0.0005180084 5.165867
# woe.balance 0.0015275894 82.164812
# woe.housing 0.0023875829 31.624682
# woe.loan -0.0004284693 18.654597
# woe.poutcome 0.0186956447 84.502653
```

上述代码中，随机森林的几个参数需要了解一下，nPerm 表示计算 importance 时的重复次数，数量大于 1 给出了比较稳定的估计；mtry 表示选择的分裂属性的个数；importance 表示输出分裂属性的重要性。平均准确率下降最小的是年度余额和是否有违约记录，平均基尼系数下降最小的是是否有违约记录和个人负债，我们剔除是否有违约记录这个变量，重新进行建模，如下所示。

- 重新建模

```
1 fit.rf <- randomForest(y~.-woe.default,data = train,ntree = 100,mtry =
10,importance = TRUE)
2 pre.forest = predict(fit.rf, test) #type='class'
3 table(pre.forest ,test$y)
# pre.forest no yes
# no 1131 133
# yes 64 27
```

设置 randomForest 的树参数为 100 棵，每棵树随机选取 10 个变量作为其节点变量选择集。从输

出结果可以看到随机森林的误判率更低，但查全率也很低。当我们想尽可能地命中更多的目标客户时，随机森林没有逻辑回归灵活。

### 6.2.4 人工神经网络建模

我们不妨顺便使用人工神经网络重复以上的任务，看看它的效率如何。

在进行神经网络建模之前需要先对数据进行标准化，我们使用 `scale` 函数，输入数据为转化为 WOE 值的变量，如下所示。

- 变量标准化

```
1 norm.data <- scale(woemodel$xnew) #神经网络需要对数据进行标准化
2 mydata.c <- data.frame(norm.data,y = mydata$y)
3 mydata.c$y <- as.factor(as.character(mydata.c$y))
```

接着我们依然按 7 : 3 的比例将数据集划分为训练集和测试集，需要特别注意的参数是隐含层神经网络节点的个数，要根据数据集不断调整出最优的参数，如下所示。

- 神经网络建模

```
1 set.seed(123) ##划分训练集和测试集
2 nr <- sample(nrow(mydata.c),nrow(mydata.c) * 0.7,replace = F)
3 train2 <- mydata.c[nr,]
4 test2 <- mydata.c[-nr,]
5 library(nnet) ## 神经网络建模
6 model.nnet <- nnet(y ~ ., ,size = 6, maxit = 1000, trace = F, data = train2 )
7 pre.forest <- predict(model.nnet, test2, type='class') #对分类数据预测需要加
#上 type 参数

8 table(pre.forest,test2$y)
# pre.forest  no  yes
#           no 1158 134
#           yes   37  26
```

从准确率上来说，在这个数据集上，神经网络比随机森林的误判率更低，当然，召回率也更低。

## 6.3 必须面对的模型评估

由于逻辑回归输出的是样本归为正样本的可能性数值，不像随机森林或者神经网络，输出的是一个 1 或者 0 的数值，因此，简单地划定一个阈值会失去很多有价值的信息。

很多资料除了用准确率这个简单粗暴的指标来衡量逻辑回归的性能外，还会用提升图或者 ROC 图来判断，但是这两个图并不是那么好理解。

我们来绘制模型的 ROC 图，看一看准确率比较高的客户集中在哪里，并根据这些分配营销资源，先将测试集上的样本按逻辑回归预测的概率值来排序，分为 10 段，每段为 10% 的样本的数据，分别

计算每一段的准确率。

- 模型的 ROC

```
1 t <- data.frame(p=p_log_test,lable=ifelse(test$y=='yes',1,0))
2 t.2 <- t[with(t, order(-p)), ]
3 top.n <- seq(0,1,0.1)##分段
4 ans <- list()
5 for(i in 1:10){
6   x0 <- ceiling(top.n[i] * nrow(t.2))
7   x1 <- ceiling(top.n[i+1] * nrow(t.2))
8   true <- sum(1 == t.2[x0:x1,2])
9   flase <- sum(0 == t.2[x0:x1,2])
10  total <- length(t.2[x0:x1,2])
11  p <- true / total
12  ans[[i]] <- c(true,flase,total,p)
13 }
14 t.3 <- do.call(rbind.data.frame,ans)
15 names(t.3) <- c("positive","negative","total","rate")
16 t.3
#   positive negative total      rate
# 1         49         87   136 0.36029412
# 2         19        117   136 0.13970588
# 3         15        122   137 0.10948905
# 4         15        121   136 0.11029412
# 5         22        115   137 0.16058394
# 6         12        125   137 0.08759124
# 7         10        126   136 0.07352941
# 8          6        130   136 0.04411765
# 9          7        130   137 0.05109489
# 10         5        131   136 0.03676471
```

从上面的结果可以看出，概率最高的前 10% 的样本的准确率达到 36%，远远大于其余 9 段的样本准确率，因此，当营销费用有限时，圈定营销客户范围可以只取这 10% 的客户。

分段统计命中率和 lift 图想要表达的含义有点类似，但是 lift 图制作得更加精细一点，它没有要求要分为几段。如果我们要标记混淆矩阵中的数值，其如图 6-7 所示。

	实际	
预测	正	负
正	<i>a</i>	<i>b</i>
负	<i>c</i>	<i>d</i>

图 6-7

那么  $\text{lift} = (d/b+d)/((c+d)/(a+b+c+d))$ ，它衡量的是与不使用模型的情况相比，模型的预测能力“变好”

了多少。不利用模型，我们只能利用“正例的比例是 $(c+d)/(a+b+c+d)$ ”这个样本信息来估计正例的比例，而利用模型之后，我们不需要从整个样本来挑选正例，只需要从我们预测为正例的那个样本的子集 $(b+d)$ 中挑选正例，这时预测的准确率为 $d/(b+d)$ 。

显然，lift 值越大，模型的运行效果越好。

- 模型的排序性曲线和 AUC 值

```
1 library(ROCR)
2 pred <- prediction(p_log_test,test$y)
3 perf <- performance(pred,"tpr","fpr")
4 plot(perf,main="ROC",ylab="TPR",xlab="FPR",col="blue")
5 auc <- performance(pred,measure = "auc")@y.values
6 text(0.5,0.2,paste("area=",auc))
```

上述代码中，我们使用 pred 和 perf 来计算模型的 ROC 曲线的 FPR 和 TPR，以及 AUC 值。结果如图 6-8 所示，模型的 AUC 达到 0.70（随机抽选情况下则为 0.5）。

- 模型的提升度

```
1 library(ROCR)
2 predictions <- prediction(p_log_test,test$y)
3 lift <- performance(predictions,measure='lift')@y.values[[1]]
4 depth <- performance(predictions,measure='rpp')@y.values[[1]]
5 plot(depth,lift,type="l",main="Lift",ylab="lift",xlab="depth",col='blue')
####depth 为 0.2
6 sep <- 100
7 for(i in 1:length(depth)){
8   x <- abs(depth[i] - 0.2)
9   if(x < sep) {
10     sep <- x
11     best_depth <- depth[i]
12     best_lift <- lift[i]}}
13 print(c(best_depth,best_lift))
14 abline(v=best_depth,col="red",lty = 2)
15 abline(h=best_lift,col="red",lty = 2)
####depth 为 0.4
16 sep <- 100
17 for(i in 1:length(depth)){
18   x <- abs(depth[i] - 0.4)
19   if(x < sep) {
20     sep <- x
21     best_depth <- depth[i]
22     best_lift <- lift[i]}}
23 abline(v=best_depth,col="red",lty = 2)
24 abline(h=best_lift,col="red",lty = 2)
```

效果如图 6-9 所示。

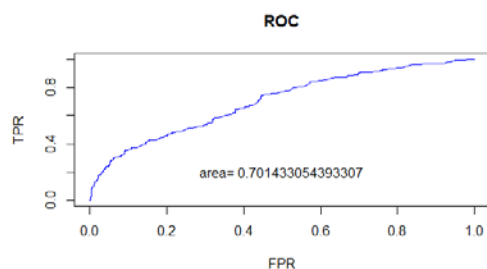


图 6-8

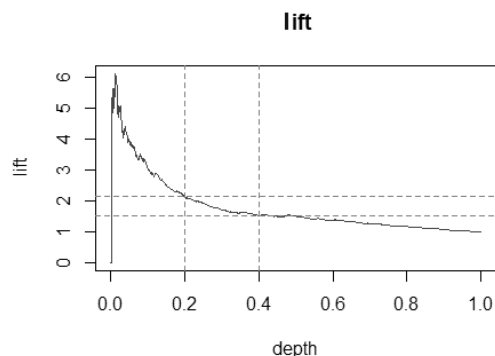


图 6-9

上述代码中，depth 表示使用了前百分之多少的样本。我们使用 ROCR 里面的 prediction 和 performance 函数来快速计算 depth 和 lift 的关系。可以看出，在 depth=0.2 时，lift=2.14，在使用模型后，我们取概率前 20% 的客户来营销相对于取全体客户来营销提升了 2.14 倍。当 depth=0.4 时，lift=1.52843，可以看出，lift 曲线的拐点是在 depth=0.4 的时候，因此，我们可以选择前 40% 的客户来进行营销。

到这里，评价模型的工具箱里除了评价分类任务的准确率和预测任务的一些统计指标，如平均绝对误差（MAE）、平均平方差（MSE）、标准平均方差（NMSE）和均值等评价指标外，又为评价分类任务的模型添加了 ROC 和提升度指标，ROC 和提升度帮助你在资源有限的情况下找到成功率较高的用户群体，集中“糖衣炮弹”笼络顾客，避免资源的浪费。

# 第 7 章

## 深入浅出十大算法

C5.0、K-means、Support Vector Machines、Apriori、EM、PageRank、AdaBoost、KNN、Naive Bayes、CART 这十大最有影响力的数据挖掘算法，是 ICDM 通过投票调研认证的封号。十大最有影响力的数据挖掘算法是每一个数据分析和挖掘人员的必备技能，你不一定能够推导出数学公式，但一定要掌握其中的关键点，同时学会在 R 中使用这几个算法能够让你了解 R 语言的优劣势。毫不夸张地说，当你掌握这几个算法之后，再看数据挖掘应该是一片山明水净的景象！现在，让我们逐一介绍这十大算法。

### 7.1 C5.0 算法

或许你听到最多的是 C4.5 决策树，但是，不是笔误，本节想要讲的就是 C5.0。决策树的概念只有一种，但其背后的算法有很多种，常见的算法有 CHAID、CART 和 C5.0。各种决策树算法大同小异，主要区别就是刻画“差异”的方式不同，但它们的主要思想都是衡量在某个类别区间上正负样本的差异程度。就好像衡量一个人的体重的时候，中国大陆用的是公斤，但中国香港用磅这个单位。但不管是公斤也好，磅也好，都只是对体重的一个刻画，并没有太大的差异。

C5.0 算法对 C4.5 算法进行了改良，占用的内存资源更少，算法计算速度更快，更加适用于处理大数据集，其采用 Boosting 方式提高模型准确率，对于每一个分支节点，要求分成的组之间的差异最大。此外，C5.0 可生成多分支的决策树，使用 C5.0 算法可以生成决策树或者规则集。

#### 7.1.1 一个重要的概念：信息熵

回到我们刚才所说的“差异”这个名词，其实有一个比较专业的说法：信息熵。什么是信息熵？信息熵是一种数学期望，衡量信息的平均不确定性，也称先验熵，是经常会用到的衡量信息的不确定性的一种指标。C5.0 算法选择分支变量的依据就是以信息熵的下降速度作为确定最佳分支变量和分割阈值的依据，因为信息熵的下降意味着信息的不确定性下降。



信息熵的定义公式：

$$H(U) = -\sum_i P(u_i) \log_2 \frac{1}{P(u_i)} = -\sum_i P(u_i) \log_2 P(u_i)$$

信息熵  $H(U)$  的性质：

- (1)  $H(U)=0$  时，表示只存在唯一的可能性，不存在不确定性。
- (2) 如果信源的  $k$  个信号有相同的发出概率，即所有的  $u_i$  有  $P(u_i)=1/k$ ， $H(U)$  达到最大，不确定性最大。

那么究竟怎样使用信息熵来生成决策树呢？首先，我们先来学习几个重要公式：设  $S$  是一个样本集合，目标变量  $C$  有  $K$  个分类， $\text{freq}(C_i, S)$  表示属于  $C_i$  类的样本数， $|S|$  表示样本集合  $S$  的样本数。则集合  $S$  的信息熵定义为：

$$\text{Info}(S) = -\sum_{i=1}^k ((\text{freq}(C_i, S) / |S|) \times \log_2 (\text{freq}(C_i, S) / |S|))$$

如果某属性变量  $T$  有  $N$  个分类，则属性变量  $T$  引入后的条件熵定义为

$$\text{Info}(T) = -\sum_{i=1}^n ((|T_i| / |T|) \times \text{Info}(T_i))$$

属性变量  $T$  带来的信息增益为

$$\text{Gain}(T) = \text{Info}(S) - \text{Info}(T)$$

C5.0 模型根据带来最大信息增益的字段拆分样本，第一次拆分整个样本集，随后再次拆分，通常是根据另一个字段进行拆分，这一过程重复进行，直到样本子集在限定的条件内不能再被拆分为止，通常限制待分割节点的样本量少于某个阈值或者达到最大决策树的规定层数。最后，C5.0 会对树中各个字段进行进一步筛选，修剪对模型值没有显著贡献的样本子集。

## 7.1.2 非列变量选择的实例

有没有觉得看了上面的公式，需要怀疑一下人生？没事，我们来实操一下，读者可以对照着上面的公式，加深一下对公式的理解。

举个例子，如表 7-1 所示。

表 7-1

	是否单身	加班情况	上班穿着	职业
T1	有女朋友	加班	t-shirt、拖鞋	程序员
T2	有女朋友	不加班	正装	霸道总裁

续表

	是否单身	加班情况	上班穿着	职 业
T3	单身	不加班	正装	程序员
T4	单身	偶尔加班	正装	程序员
T5	单身	加班	正装	程序员
T6	单身	加班	正装	霸道总裁
T7	单身	加班	t-shirt、拖鞋	程序员
T8	有女朋友	偶尔加班	正装	霸道总裁
T9	单身	加班	正装	程序员
T10	有女朋友	不加班	正装	霸道总裁

假设数据集有三个预测变量：是否单身，加班情况，上班穿着；一个目标变量：职业。由于决策树算法是递归的，所以我们只计算哪个变量会成为决策树的根节点，建议自行计算剩下的部分，以加深对算法的理解。

$$\text{Info(职业)} = -6/10 \log_2(6/10) - 4/10 \times \log_2(4/10) = 0.9709506$$

是否单身中单身一共有 6 个样本，则：

$$\text{info(单身)} = -5/6 \times \log_2(5/6) - 1/6 \times \log_2(1/6) = 0.6500224$$

$$\text{info(有女朋友)} = -1/4 \times \log_2(1/4) - 3/4 \times \log_2(3/4) = 0.8112781$$

$$\text{因此：Info(是否单身)} = 6/10 \times \text{info(单身)} + 4/10 \times \text{info(有女朋友)} = 0.7145247$$

$$\text{同理，Info(加班情况)} = 5/10 \times (-4/5 \times \log_2(4/5) - 1/5 \times \log_2(1/5)) + 2/10 \times (-1/2 \times \log_2(1/2) - 1/2 \times \log_2(1/2)) + 3/10 \times (-1/3 \times \log_2(1/3) - 2/3 \times \log_2(2/3)) = 0.8364528$$

$$\text{Info(穿着情况)} = 2/10 \times (-\log_2(1)) + 8/10 \times (-4/8 \times \log_2(4/8) - 4/8 \times \log_2(4/8)) = 0.8$$

三个变量的信息增益为：

$$\text{Gain(是否单身)} = 0.9709506 - 0.7145247 = 0.2564259$$

$$\text{Gain(加班情况)} = 0.9709506 - 0.8364528 = 0.1344978$$

$$\text{Gain(穿着情况)} = 0.9709506 - 0.8 = 0.1709506$$

因此，是否单身这个信息带来的信息增益最大，选择是否单身来作为分裂变量。

### 7.1.3 C5.0 算法的 R 实现

我们使用 R 中自带的鸢尾花数据集来实现一下 C5.0 算法，为了考察各种算法在同一数据集上的表现，对于后面即将讲到的算法，我们尽量使用同一个数据集进行测试，虽然这种方法比较不具有统计意义。

- 拆分数据集

```
1 library("C50")
2 train.idx <- sample(1:nrow(iris), 100)
```

```
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

上述代码中，因为鸢尾花数据集总共包括 150 行数据，我们将其分为训练集和测试集，sample 函数从 1~150 中随机选择 100 个数字，这些随机选择的数字将组成训练集的行编号保存在 train.idx 变量里；第 3 行代码把刚才随机选择的数字对应到行号，选择相应的样本（逗号后面留空意味着所有列都被选择了），保存在 iniris.train 数据框中；第 4 行代码表示选择剩余的样本，同样，所有的样本属性都被包含在里面，保存在 iris.test 数据框中，预留为测试集。这样我们就简单地将数据分为了训练集和测试集，如果你需要将数据分成用于交叉检验的分组，具体代码和思路请参见第 10 章。

我们怎么训练 C5.0 呢？训练是算法最复杂的部分，但你只需要用 1 行 R 代码就可以完成了，如下所示。

- 训练和测试

```
1 modelc5 <- C5.0(formula = Species ~ ., data = iris.train, trials = 100,
rules = TRUE)
2 resc5 <- predict(object = modelc5, newdata =iris.test, type ="class")
```

上述代码中，第 1 行代码使用 C5.0 函数构建模型，第 1 个参数用于指定模型的因变量和自变量，中间用“~”分开，第 2 个参数用于指定数据集，trials 参数用于指定迭代的次数，rules 选择是否生成规则集。

第 2 行代码使用 predict 函数测试刚刚训练的模型。class 参数告诉模型是要进行分类（本例中就是要区分鸢尾物种），把参数都正确代入后，分类器就会在测试数据的预测变量里把鸢尾分好类，结果保存在 resc5 变量里。

怎么检查是否正确分类了呢？一个快速的方法就是生成一个混淆矩阵。混淆矩阵也叫列联表，它可以直观地告诉我们分类的情况，包括正确的分类、错误的分类。

- C5.0 模型结果

```
1 table(iris.test$Species, resc5)
#           resc5
#           setosa versicolor virginica
#   setosa         13          0          0
#   versicolor      0          20         2
#   virginica       0          0         15
2 table(iris.test$Species)
# setosa versicolor virginica
#      13         22         15
```

混淆矩阵的列表示模型预测的鸢尾花的物种，行表示正确的物种分类（具体看你 table 函数的参数顺序）。上述代码中，可以看到模型对 setosa、virginica 的分类效果最好，而 versicolor 有两个被错分成了 virginica，按比例来说模型对 versicolor 分类最差；可以通过 table 函数计算测试集中三种鸢尾花的数量。

这里仅仅是做了一次简单的对比，如果要进行检验和比较需要进行召回率和准确率的计算，并进行交叉检验，可以参看第 10 章。

## 7.2 K-means 算法

K-means 算法是一种应用最广泛的聚类算法，是一种基于划分的聚类方法。算法的基本思想是：以空间中  $K$  个点为中心进行聚类，对最靠近它们的点进行归类。通过迭代的方法，逐次更新各聚类中心点的值，直至得到最好的聚类结果，通常为中心点的位置不再因迭代次数的增加而变化。同时，这使所获得的聚类满足：同一聚类中的对象相似度较高，不同聚类中的对象相似度较小。当然衡量这个定义的指标比较多，比如轮廓系数等基本上都是基于这个目的构建的衡量聚类效果的指标。

K-means 算法假设要把样本集分为  $c$  个类别，算法描述如下：

- (1) 随机选择  $c$  个类的初始中心；
- (2) 在第  $n$  次迭代中，对任意一个样本，求其到每一个中心的距离，将该样本归到距离最近的中心所在的类；
- (3) 更新该类的中心值，一般利用均值、中位点等方法；
- (4) 对于所有的  $c$  个聚类中心，利用 (2)(3) 的迭代法更新后，如果中心点的值不再变化，则迭代结束，否则继续迭代，当达到最大迭代次数时，中心点仍未收敛的，取最后的中心点为中心。

算法的关键在于初始中心的选择和计算样本到中心距离的定义公式。由于 K-means 容易收敛到局部最优值，因此，初始点不同，最后的聚类结果可能会不一样，不要觉得意外，很多内部包含随机过程的算法每次输出的结果都会出现一些微小差异，但可以选取多次不同的初始值，反复使用 K-means，使得最后收敛的中心点出现的次数最多。另外距离公式主要依据现实世界对差异的定义，也就是说衡量距离的方法比较多，根据业务和数据类型选择合适的距离计算方法，一般情况下距离定义为欧式距离。

该算法的最大优势在于简洁和快速，而且是一个非监督的过程。

### 7.2.1 K-means 算法的 R 实现

我们仍然使用鸢尾花数据集实现 K-means 算法。

```
1 require("stats")
2 model <- kmeans(x = subset(iris, select = -Species), centers = 3)
3 table(iris$Species, model$cluster)
#           1  2  3
# setosa     50  0  0
# versicolor  0 48  2
# virginica   0 14 36
```

同样，只要一行代码，我们就可以搭建好 K-means 算法了（见第 2 行代码）。这一行代码主要做了以下两件事：① subset 函数用于将鸢尾花的物种列从数据集中移除，对于非监督模型我们没有必要保留目标变量；② 调用 kmeans 函数，代入去掉了物种列的数据集，以及把聚类的类别数设为 3。好多读者分不清楚 kmeans 函数几个参数的具体作用：centers 设置聚类数；iter.max 设置最大迭代次数；nstart 随机设置初始中心点的次数，也可以理解为重复做 kmeans 的次数，众所周知，kmeans 对

初始中心非常敏感, `nstart` 为 10 表示做 10 次 `kmeans`, 每次初始中心点随机生成一次, 也可以理解为确定初始中心点时重复 `kmeans` 的次数, 每次迭代完成后返回最优解, 当 `nstart` 等于 1 时, 等于只做了一次 `kmeans`, 由于初始中心点不同, 所以每次结果都不一样 (可以设置随机种子克服), `nstart` 一般设置在 20~25 之间, 基本保证能返回最优解, 每次的聚类结果也大体相同, 除非你的数据量很大。

使用 `table` 函数制作混淆矩阵, 发现 `kmeans` 将 `setosa` 聚在了第 1 类, 第 2 类主要是 `versicolor`, 但将 14 个 `virginica` 也分到了第 2 类, 整体上效果比较一般。

## 7.2.2 怎么确定聚类数

聚类就像切西瓜, 切几瓣真的很随意, 但是我们也可以使用一些方法评估一下应该聚几类, 比如轮廓系数, 它结合了内聚度和分离度两种因素, 即同时考察了组内相似性和组间差异性, 可见轮廓系数的值介于  $[-1, 1]$ , 越趋近于 1 代表内聚度和分离度都相对较优, 我比较习惯使用这个指标。

- 轮廓系数

```
1 library(cluster)
2 asw <- numeric(20)
3 for (k in 2:20) {
4   asw[[k]] <- pam(subset(iris, select = -Species), k)$silinfo$avg.width
5 }
6 k.best <- which.max(asw)
7 cat("通过轮廓系数 (silhouette) 估计的最佳聚类数:", k.best, "\n")
```

上述代码中, 第 2 行代码生成一个序列; 第 3 行代码测试聚类数 2~20 的可能性, 使用循环调用 `pam` 函数构建聚类, 并提取结果中的 `avg.width` 赋值给相应的 `asw`; 第 4 行代码找出 `asw` 最大值所对应的聚类数; 第 7 行代码使用 `cat` 函数输出最佳聚类数。

你也可以使用中心化分法 (PAM) 确定聚类的数量, 如下所示。

- 中心化分法 (PAM)

```
1 library(fpc)
2 library(cluster)
3 pamk.best <- pamk(subset(iris, select = -Species))
4 cat("通过中心化分法估计的最佳聚类数:", pamk.best$nc, "\n")
```

`pamk` 函数来自 `fpc` 包, 用于衡量聚类的  $K$  值, 即聚类数。很遗憾, 两种方法给出的最佳聚类数都是 2, 很明显告诉我们选择这种非监督模型不是一个好的思路。虽然  $K$ -means 算法在这个案例里确实表现得并不好, 但是, 很遗憾地告诉你, 并没有哪个算法能在任何的数据里都表现得很好, 更加遗憾的是我们并不一定能够找到准确描述预测数据的方法。

## 7.3 支持向量机 (SVM) 算法

支持向量机这个名字确实很不形象, 很难从这个名字联想到这个算法的作用, 那么我们先来形

象地认识一下这个算法。

### 7.3.1 通俗理解 SVM

在空间中有两种点，黑点和灰点，如图 7-1 所示，现在要用一条直线把它们分隔开。显然，你马上知道有无数条分隔线，可以把这两种点完美地分隔，如图 7-2 所示。

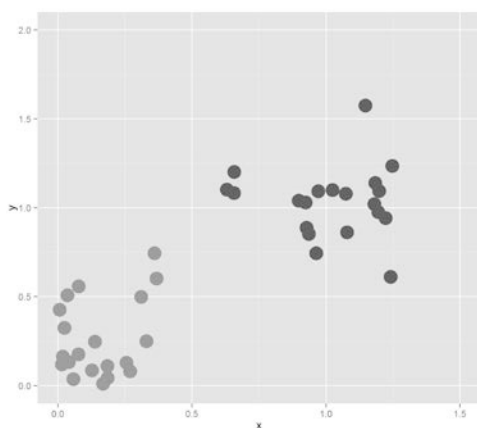


图 7-1

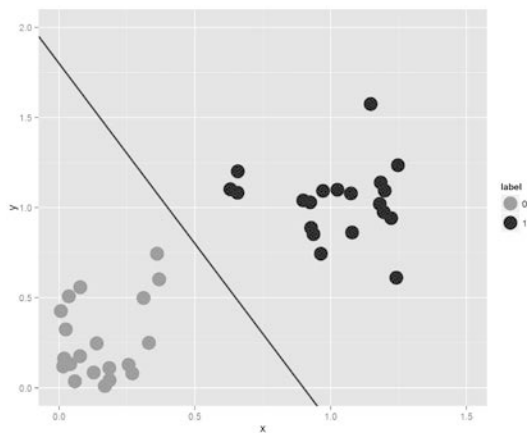


图 7-2

那么问题来了，选择哪条分隔线呢？SVM 就试图把线放在最佳位置，好让线的两边有尽可能大的间隙，如图 7-3 所示。

显然，如果空间上有更多的点，处于最佳位置的分隔线仍然是一个好的分界线，如图 7-4 所示。

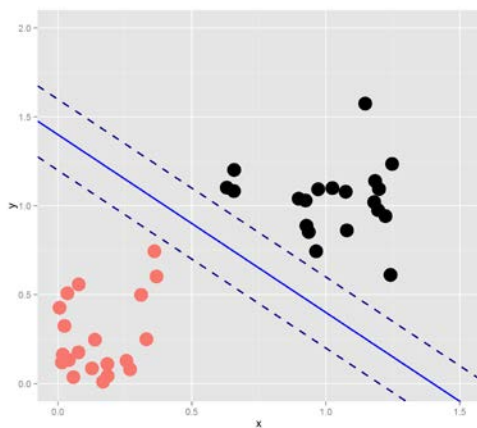


图 7-3

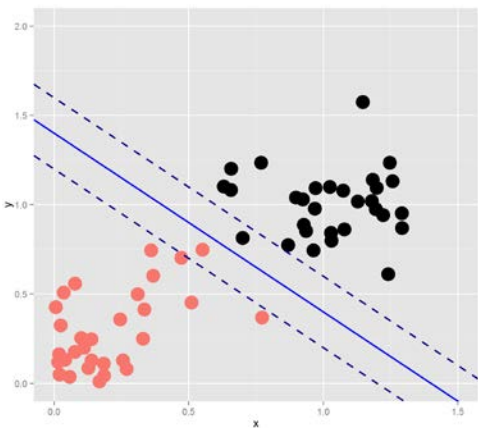


图 7-4

在线性可分的情况下，SVM 已经做到了极致，那么如果线性不可分呢？如图 7-5 所示。

现在，如果要分开空间中的这两种点，怎么办呢？大概那条分隔线会这样吧，如图 7-6 所示。

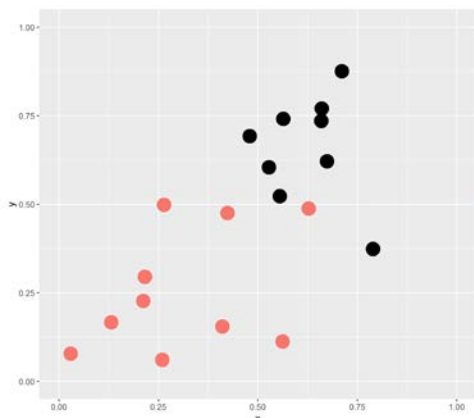


图 7-5

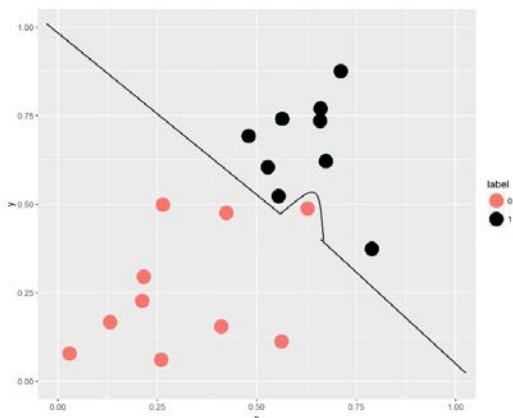


图 7-6

乍看起来只有曲线才能解决这种问题，但其实，SVM 可以通过将低维线性不可分的情况转化为高维线性可分来处理。你猜得没错，这就是核的思想。

如图 7-7 所示，通过把二维平面上线性不可分的点转化到三维空间，从而达到用一个平面分隔不同类别的点的作用。

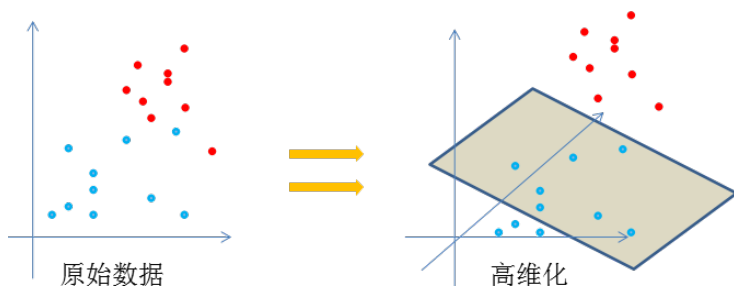


图 7-7

看完上面的解释，相信大家都知道支持向量机的思想了，但又为什么要叫支持向量机呢？支持向量指的是数据集中的某些点，我们知道，支持向量机只关注最靠近划分直线的那几个点，而样本集中的其他远离决策平面的点对决策平面的最终位置的确定起不了作用。也就是决定支持向量机决策边界的点只是这些边界点，所以把这些点称为“支持向量”。“机”的意思是算法，机器学习领域里面常常用“机”这个字表示算法，例如受限波尔兹曼机、有限状态机，所以就叫支持向量机了。

## 7.3.2 SVM 的 R 实现

我们训练 SVM 模型对鸢尾花进行分类，然后测试 SVM 模型在鸢尾花分类任务上的准确率。

- 数据集分组

```
1 library("e1071")
2 train.idx <- sample(1:nrow(iris), 100)
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

上述代码用来生成测试集和训练集。就像 C5.0 算法一样，训练也是这个算法里面最复杂的部分，但是，我们仍然只需要用一行代码就可以完成这个艰巨的任务，如下所示。

- 构建 SVM 模型

```
1 modelsvm <- svm(Species ~ ., data = iris.train, kernel = "radial")
```

就像训练 C5.0 分类器一样，svm 函数的第 1 个参数用于指定数据之间的函数关系，即哪个是目标变量，哪些是自变量，它们之间用“~”号分开，英文句号表示数据集中除目标变量外的所有变量都加入到模型；第 2 个参数指定数据集；第 3 个参数 kernel 用于指定核函数。

模型构建完成后，我们接着简单测试一下模型，代码如下。

- 模型测试

```
1 ressvm <- predict(object = modelsvm, newdata = iris.test, type = "class")
2 table(iris.test$Species, ressvm)
#           ressvm
#           setosa versicolor virginica
# setosa          16           1         0
# versicolor       0          15         0
# virginica        0           1        17
```

结果怎么样呢？同样生成一个混淆矩阵，SVM 分类器表现得不错。虽然目前为止所做的测试都没有很深入，但从混淆矩阵的结果来看，SVM 和 C5.0 在鸢尾花数据集上表现一样好，比 K-means 聚类好得多。

至于 SVM 算法中核函数的选择方法，可以参看第 10 章。

## 7.4 Apriori 算法

或许你不知道 Apriori 这个名词，但我相信大部分读者都知道啤酒与尿布这个故事，这个故事的大意是：在超市购物时，有一个奇特的现象，顾客在买完尿布之后通常会买啤酒，即{尿布}→{啤酒}。因为妻子嘱咐丈夫回家的时候给孩子买尿布，丈夫买完尿布后通常会买自己喜欢的啤酒。销售人员发现这个现象之后，就把啤酒和尿布放在一起，以期待增加啤酒的销量。先不管这个故事是不是像



牛顿和苹果的故事那样是个假故事，反正这个故事所体现的核心思想就是本节要阐述的。Apriori 是一种关联分析算法，其中最有名的问题是购物篮问题。

Apriori 算法的核心是基于两阶段频集思想的递推算法。首先我们来认识几个概念：支持度、可信度。支持度即不同物品同时出现的频次，可信度即一个物品出现则另一个物品也会出现的概率，支持度是一个相关的概念，而可信度更像是一种因果关系。其次我们要定义一个最小支持度、一个最小可信度。最小支持度即我们认为两个物品具有高度相关性最小应该出现的频次，最小可信度即一个物品出现，另外一个物品同时也出现的概率下限。在这里，所有支持度大于最小支持度的项集称为频繁项集，简称频集。

该算法的基本思想是：首先找出所有的频集，这些项集出现的频繁性至少和预定义的最小支持度一样，然后由频集产生强关联规则，这些规则必须满足最小支持度和最小可信度。其中每一条规则的左边可以有多项，而右边只有 1 项，使用递归的方法生成所有频集。简单点说就是先找频繁项集只有 1 项的元素，在这基础上找频繁项集有 2 项的元素，以此类推，可以这样来表示：在频繁项集有  $K$  项的元素中找频繁项集有  $K+1$  项的元素。

### 7.4.1 举例说明 Apriori

下面我们通过一个例子，外加一点点的计算来加深我们对 Apriori 算法的理解，如表 7-2 所示。

表 7-2

单 据	项 目
A1	啤酒, 尿布, 牛奶
A2	尿布, 巧克力
A3	尿布, 苹果
A4	啤酒, 尿布, 巧克力
A5	啤酒, 苹果
A6	尿布, 苹果
A7	啤酒, 苹果
A8	啤酒, 尿布, 苹果, 牛奶
A9	啤酒, 尿布, 苹果
A10	麦片

假设表 7-2 是某商场购物系统上的记录，每一行表示一张超市购物小票上的购物项，左边表示购物小票的唯一 ID，右边表示商品名称。

我们先来设置最小支持度，例如我们设置为 20%，那么一件商品如果符合频繁项定义，那它至少要出现 2 次。首先进行第一次数据库扫描，计算每个商品单独出现的次数，之后我们第二次扫描数据，找频繁项集项个数为 1 的元素，如表 7-3 所示。

表 7-3

商品名称	出现次数
啤酒	6
尿布	7
苹果	6
巧克力	2
牛奶	2
麦片	1

左边表示商品名称，右边表示出现的次数，除了 I6，其他项都大于阈值 2，我们把 I6 过滤掉。在此基础上我们再找频繁项集是 2 的元素，方法是两两任意组合，第三次扫描数据得到它们出现的次数，如表 7-4 所示。

表 7-4

商品名称	出现次数
啤酒, 尿布	4
啤酒, 苹果	4
啤酒, 巧克力	1
啤酒, 牛奶	2
尿布, 苹果	3
尿布, 巧克力	2
尿布, 牛奶	2
苹果, 巧克力	0
苹果, 牛奶	1
巧克力, 牛奶	0

可以知道，{I1,I4},{I3,I4},{I3,I5},{I4,I5}出现的次数都小于 2，过滤掉，实际频繁项集为 2 的元素如表 7-5 所示。

表 7-5

商品名称	出现次数
啤酒, 尿布	4
啤酒, 苹果	4
啤酒, 牛奶	2
尿布, 苹果	3
尿布, 巧克力	2
尿布, 牛奶	2

在此基础上找频繁项集为 3 的元素，此时就有规律性了，在频繁项集为  $K$  的元素上找频繁项集为  $K+1$  的元素的方法是：在频繁项集为  $K$  的项目（每行记录）中，假如共有  $N$  行，两两组合，满足两两中前  $K-1$  个元素相同，最后一个元素要求前一条记录的商品编号小于后一条记录的商品编号，这样是为了避免重复组合，并求它们的并集得到长度为  $K+1$  的准频繁项集，那么最多共有  $C_N^K$  种。想想如果  $N$  很大， $C_N^K$  是一个多么庞大的数字，这时就要用到 Apriori 的核心了：如果  $K+1$  个元素构成频繁项集，那么它的任意  $K$  个元素的子集也是频繁项集。将每组  $K+1$  个元素的所有长度为  $K$  的子集（有  $C_{K+1}^K$ ）种组合，在频繁项集为  $K$  的项集中匹配，没有找到则删除，用第一条记录{啤酒,尿布,牛奶}，它的长度为 2 的频繁项集有  $C_3^2=3$ ，分别是：{啤酒,尿布},{啤酒,牛奶},{啤酒,牛奶}

通过这步过滤，得到的依旧是准频繁项集，它们是：{啤酒,尿布,苹果},{啤酒,尿布,巧克力},{啤酒,尿布,牛奶}，此时第 4 次扫描数据库，得到真正长度为 3 的频繁项集，如表 7-6 所示。

表 7-6

商品名称	出现次数
啤酒, 尿布, 苹果	2
啤酒, 尿布, 巧克力	2

因为{啤酒,尿布,牛奶}只出现了 1 次，小于最小支持度 2，所以将其删除。

就这个例子而言，它的最大频繁项集只有 3，就是{啤酒,尿布,苹果}和{啤酒,尿布,巧克力}

可能产生大量的候选集，以及可能需要重复扫描数据库，是 Apriori 算法的两大缺点。针对 Apriori 的缺点，又产生了 FPTree 算法，它可以在不生成候选项的情况下，完成 Apriori 算法的功能，算法的时间复杂度比 Apriori 算法快一个数量级，在空间复杂度方面也比 Apriori 有数量级级别的优化。

## 7.4.2 Apriori 算法的 R 实现

我们在第 9 章使用 Apriori 算法预测演员之间的关系，讲解得比较详细，大家可以直接模仿那一章的数据录入，我们这里仅仅使用 arules 包自带的演示一下 Apriori 算法。使用 Apriori 算法来挖掘人口普查的收入数据，目的是发现调查数据集里的相关项，相关项也叫项集，项集里的关系就叫关联规则，而我们的任务就是找出其中有趣的规则并运用到实际场景中。现在，让我们开始运用 Apriori 算法！使用 Apriori 算法时，在 R 中有多种数据摆放形式，你可以参考说明文档，例如 read.transactions 函数使用 single 参数进行数据整形，也可以参看第 9 章。

### • 关联分析

```
1 require("arules")
2 data("Adult")
3 rules <- apriori(Adult, parameter = list(support = 0.4, confidence = 0.7),
appearance = list(rhs = c("race=White", "sex=Male"), default = "lhs"))
```

第 1 个参数设定数据集为 Adult 数据集，并将其关联规则储存在 rules 里面。parameter 设定支持度、置信度参数过滤掉不感兴趣的规则，appearance 给 apriori 函数做一些字符设定，来寻找特定的

关联规则。关联规则在总体  $D$  中的支持度 (support) 是  $D$  中事务同时包含  $X$ 、 $Y$  的百分比，即概率，这里把支持度设定为 40%。置信度 (confidence) 是  $D$  中事务已经包含  $X$  的情况下，包含  $Y$  的百分比，即条件概率。我们在这个案例里使用 70% 的置信度，一般置信度要在 75% 以上。

如果满足最小支持度阈值和最小置信度阈值，则认为关联规则符合要求。支持度和置信度都不宜太低，如果支持度太低，则说明规则在总体中占据的比例较低，缺乏商业价值，如果置信度太低，则很难从  $X$  关联到  $Y$ ，同样不具有商业价值。事实上，Apriori 算法可以发现一大堆的关联规则，你需要用支持度和置信度来过滤出一些你感兴趣的规则。关联规则类似这样： $\{\text{United States}\} \Rightarrow \{\text{White, Male}\}$ ，它表示“当出现美国时，出现白人男性的支持度、置信度”，这里需要告诉函数使用左向规则 (lhs) 或右向规则 (rhs)。

- 找出关联规律

```
1 rules.lift <- sort(rules, by = "lift")
2 inspect(rules.lift[1:5])
3 top5.rules <- head(rules.lift, 5)
4 as(top5.rules, "data.frame")
```

上述代码中，第 1 行代码根据提升度将规则排序，提升度表示含有  $X$  的条件下，同时含有  $Y$  的概率，与不含  $X$  的条件下却含  $Y$  的概率之比，提升度告诉我们关联规则的左边和右边关联在一起的强度，提升度越高，关联规则越强；第 2 行代码查看提升度排序前 5 名的规则；第 3 行代码选取提升度最高的 5 条规则；第 4 行代码选取提升度最高的 5 条规则，再把它们转化为一个数据框，以便于我们展示它们。

前 5 条规则说明了什么呢？第 1 条规则，当我们看到“husband”这个关系时，几乎可以肯定应该是个男人，这没什么奇怪的，令人奇怪的是这条规则的置信度不是 100%，原因你懂的。第 2 条规则，它和第 1 条规则类似，在这里只是加了“civilian spouses”项的限制条件。第 3 条和第 4 条规则，我们看到在“civilian spouse”（配偶非公职人员）中，我们有很大机会看到男性和白种人，即配偶为非公职人员的人群中多数是男性或白种人。这是一条有价值的信息，因为它暗示我们关于数据的某些分布。为什么不是女性呢？为什么不是其他肤色的人种呢？

规则就在那里，你来或不来就是那个样子，但设定支持度、置信度、提升度却是一门艺术，需要具有业务的视角和逻辑才能产生出具有商业价值的规则，而使用一些不具有商业意义的规则决策至少会给你造成资源的浪费。

## 7.5 EM 算法

最大期望算法 (EM) 是一种迭代算法，通常用于含有隐变量的概率参数模型的最大似然估计或极大后验概率估计。注意关键词：隐变量。什么是隐变量呢？举个例子，假设现在有 100 人的体重数据，为 54kg、58 kg、70 kg、65 kg、44kg 等，不出意外肯定是男生或者女生组成的这 100 个人，那么对于样本体重为 54 公斤的人，我们没办法知道是男生还是女生，这其中男女就是一个隐变量，

我们只能看到 54 kg，但是看不到背后男女这个隐变量。在统计计算中，最大期望算法是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐藏变量。最大期望算法经过两个步骤交替进行计算：

(1) 初始化分布参数

(2) 以下两个步骤不断交替进行，直到收敛。

- E 步骤：利用对隐藏变量的现有估计值，计算其最大似然估计值。
- M 步骤：最大化在 E 步骤上求得的最大似然值来计算参数的值。

### 7.5.1 举例说明 EM 算法

我们通过一个例子来解释这个算法的计算和思想，方便读者理解。考虑一个投掷硬币的实验：现在我们有两枚硬币 A 和 B，这两枚硬币和普通的硬币不一样，它们投掷出正面的概率和投掷出反面的概率不一定相同。我们将 A 和 B 投掷出正面的概率分别记为  $\theta_A$  和  $\theta_B$ 。现在独立地做 5 次试验：随机地从这两枚硬币中抽取 1 枚，投掷 10 次，统计出现正面的次数，那么我们就得到了如表 7-7 所示的实验结果。

表 7-7

试验代号	投掷的硬币	出现正面的次数
1	B	5
2	A	9
3	A	8
4	B	4
5	A	7

在这个实验中，我们记录两组随机变量，第 1 组代表这次试验投掷的是硬币 A 还是硬币 B，第 2 组代表试验中出现正面的次数，如表 7-7 所示。我们的目标是通过这个实验来估计  $\theta = (\theta_a; \theta_b)$  的数值。这个实验中的参数估计就是有完整数据的参数估计，这是因为我们不仅知道每次试验中投掷出正面的次数，还知道每次试验中投掷的是硬币 A 还是 B。

一个很简单也很直接的估计  $\theta$  的方法如以下公式所示：

$$\theta_a = \frac{\text{投掷硬币 A 出现正面的次数}}{\text{投掷硬币 A 的次数}} \quad (1)$$

$$\theta_b = \frac{\text{投掷硬币 B 出现正面的次数}}{\text{投掷硬币 B 的次数}} \quad (2)$$

实际上，这样的估计就是统计上的极大似然估计的结果。用  $P(X; Z_j)$  来表示  $X, Z$  的联合概率分布，那么对于上面的实验，我们可以计算出观察到的结果。即  $X_0 = \{5, 9, 8, 4, 7\}$ ； $Z_0 = \{B; A; A; B; A\}$  的概率。令函数  $P(X = x^{(0)}, Z = z^{(0)} | \theta)$  叫作  $\theta$  的似然函数。对  $\theta$  求偏导并令

偏导数为 0，就可以得到  $\theta$  的求解结果，如下。

$$P(X = x^{(0)}, Z = z^{(0)} | \theta) = C_5^3 P(Z = A)^3 (1 - P(Z = A))^2 \times C_{10}^9 \theta_A^9 (1 - \theta_A) \times C_{10}^8 \theta_A^8 (1 - \theta_A)^2 \times C_{10}^7 \theta_B^5 (1 - \theta_B)^3 \times C_{10}^4 \theta_B^4 (1 - \theta_B)^6$$

将这个问题稍微改变一下，我们将所观察到的结果修改一下：

我们现在只知道每次试验有几次投掷出正面，但是不知道每次试验投掷的是哪个硬币，也就是说我们只知道表 7-7 中的第 1 列和第 3 列，这个时候我们就称  $Z$  为隐藏变量， $X$  为观察变量，再来估计参数  $\theta_A$  和  $\theta_B$ ，就没有那么多数据可供使用了，这个时候的估计叫作不完整数据的参数估计。如果我们这个时候有某种方法（比如，正确地猜到每次投掷硬币是 A 还是 B），这样我们就可以将这个不完整的数据估计变为完整数据估计。当然如果没有方法来获得更多的数据，那么下面提供了一种在这种不完整数据的情况下估计参数  $\theta$  的方法。我们用迭代的方式进行：

（1）我们先赋给  $\theta$  一个初始值，这个值不管是经验也好猜得也好，反正我们给它一个初始值。在实际使用中，这个初始值往往是由其他算法的结果给出的，当然随机给它分配一个符合定义域的值也可以。这里我们就给  $\theta_A = 0.7$ ； $\theta_B = 0.4$ 。

（2）然后我们根据这个来判断或者猜测每次投掷更像是哪枚硬币投掷的结果。比如对于试验 1，如果投掷的是硬币 A，那么出现 5 个正面的概率为  $C_1^5 \times 0.7^5 \times (1 - 0.7)^5 = 0.1029$ ；如果投掷的是硬币 B，出现 5 个正面的概率为  $C_1^5 \times 0.4^5 \times (1 - 0.4)^5 = 0.2007$ ；基于试验 1 的试验结果，可以判断这个试验投掷的是硬币 A 的概率为  $0.1029 / (0.1029 + 0.2007) = 0.3389$ ，是硬币 B 的概率为  $0.2007 / (0.1029 + 0.2007) = 0.6611$ 。因此这个结果更可能是投掷硬币 B 出现的结果。

（3）假设上一步猜测的结果为 B,A,A,B,A，那么根据这个猜测，可以像完整数据的参数估计一样（公式②）重新计算  $\theta$  的值。这样一次又一次地迭代（2）（3）步骤，直到收敛，我们就得到了  $\theta$  的估计。

## 7.5.2 EM 算法的 R 实现

在数据挖掘的框架中，EM 算法被运用在聚类（比较像 K-means 算法）或对齐问题当中。我们运用 EM 算法来对鸢尾花进行分类，如下所示。

### • EM 算法

```
1 library("mclust")
2 model <- Mclust(subset(iris, select = -Species))
```

我们把物种那一列去掉，就像 K-means 一样，因为都是完成聚类这个任务，只是我们这次是用 mclust 函数来做聚类。mclust 函数和 EM 算法是怎么联系起来的呢？mclust 函数的引擎正是运用了 EM 算法。简而言之，mclust 使用 EM 算法来改善一组模型，然后选择 BIC 值最低的一个。BIC 即贝叶斯信息准则，简而言之，BIC 是衡量一个模型的解释能力和模型的简单性的指标，模型越简单，以及能解释越多的数据，BIC 值就越低。

我们怎么测试 EM 聚类呢？我们可以用在 K-means 算法中使用的方法来测试这个聚类算法是否有效，如下所示。

- EM 分类测试

```
table(iris$Species, model$classification)
```

EM 算法找到了两个簇，分别用数字 1 和数字 2 来进行编号。Mclust 模型很有效地把 setosa 类从另外两个类别中区分开来。但是就像 K-means 算法一样，EM 算法难以区分 versicolor 和 virginica 类别。从上面的分析可以看出，非监督模型的准确率暂时低于监督模型。

## 7.6 PageRank 算法

PageRank 算法是谷歌搜索引擎的核心算法，该算法是由 Larry Page 和 Sergey Brin 在斯坦福大学读研时设计的，PageRank 的核心思想有两点。

(1) 如果一个网页被很多其他网页链接，说明这个网页比较重要，也就是 PageRank 值会相对较高；

(2) 如果一个 PageRank 值很高的网页链接到一个其他的网页，那么被链接到的网页的 PageRank 值会相应地因此提高。

这个算法可以引申到社交关系中的核心人物发现。例如在微博关注中，如有一个大 V 关注了一个微博账号，那么可以知道这个微博账号很有可能也是一个大 V。

- 生成随机网络关系

```
1 library("igraph")
2 library("dplyr")
3 gnet <- random.graph.game(n = 10, p.or.m = 1/4, directed = TRUE)
4 plot(gnet, edge.arrow.size = 0.01, vertex.size = 9, vertex.label.dist = 0.55)
```

PageRank 算法能发现一系列节点里相对重要的节点，我们不需要监督训练 PageRank，在测试 PageRank 之前，我们先使用 random.graph.game 生成一个随机网络关系（见图 7-8），这里是一个随机过程，所以你执行上述代码获得的图可能和本书中的不一样。另外，我们在第 9 章使用 PageRank 算法评估各国的联盟关系，对于社交网络方面的知识不了解的读者可以直接阅读。

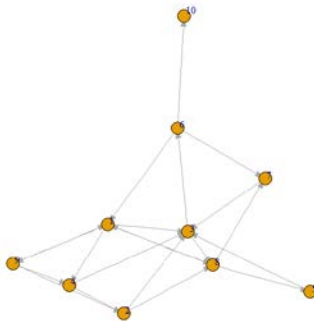


图 7-8

那么怎么将 PageRank 算法应用到这个关系网络中呢？只需要一行代码，你就可以将 PageRank 算法应用到你刚才生成的关系网络中，如下所示。

- 模型结果

```
1 pr <- page.rank(gnet)$vector
2 df <- data.frame(Object = 1:10, PageRank = pr)
3 arrange(df, desc(PageRank))
#   Object   PageRank
# 1      3 0.23159783
# 2      8 0.16526354
# 3      6 0.11792641
# 4      1 0.11200220
# 5      7 0.09973449
# 6      5 0.06632201
# 7      4 0.05556551
# 8      2 0.05538039
# 9     10 0.05290982
# 10     9 0.04329780
```

只需要一行 R 代码就调用了 PageRank 算法，它检索了网络中 10 个节点 pr (PageRank) 值向量 (见第 1 行代码)。当然，我们也可以把这个向量看成一个列表，那么我们检索的就是一个 pr 值的列表。第 2 行和第 3 行代码将 PageRank 的结果转化为数据框，并使用 arrange 函数进行了降序排列，pr 值越大代表节点在网络中越重要。如果想深入了解 PageRank 算法，请移步到第 9 章。

## 7.7 AdaBoost 算法

AdaBoost 的意思是自适应增强 (Adaptive Boosting 的缩写)。它的自适应在于：如果某个样本被一个弱分类器分错，那么在下一轮训练中，这个样本会被重视，其权重会被加强。同时将这个分类器加入到模型集合中，这样每一轮中会加入一个新的更加准确的弱分类器，直到模型达到我们所能容忍的错误率或达到我们所能容忍的最大迭代次数。

具体说来，整个 Adaboost 迭代算法就 3 步：

(1) 初始化样本的权重：假如有  $N$  个样本，则每一个训练样本最开始时都被赋予相同的权重： $1/N$ 。

(2) 重复以下步骤，直到达到停机条件：首先训练弱分类器，并不断更新样本权重。权重更新的依据如下：如果某个样本点被正确地分类，那么在构造下一个训练集中，它的权重就被降低；反之，如果某个样本点没有被正确地分类，那么它的权重就得到提高，然后，权重更新过的样本集被用于训练下一个分类器。分类器一般为同一种分类器，整个训练过程按照上述的依据，不断迭代地进行下去，直到达到预设的最大分类器个数。



(3) 将第(2)步训练得到的一系列弱分类器组合起来,同时对预测样本进行投票。根据每个弱分类器的准确率赋予每个弱分类器不同的权重,准确率高的分类器权重较大,其在最终的投票中起着较大的决定作用,而准确率低的分类器权重较小,其在最终的投票中起着较小的决定作用,这样,一系列弱分类器组合成强分类器。

AdaBoost 生成的弱分类器只有输入的样本是不一样的,并没有在其他地方做限制,使用最多的 AdaBoost 算法是使用决策树作为弱分类器的。或许你听说过随机森林算法,但随机森林算法并不是 AdaBoost,尽管它也生成多个弱分类器,但每个弱分类器所使用的变量是不一样的,这是随机森林和以决策树为弱分类器的 AdaBoost 最大的不同之处。

据说在 Deep Learning (深度学习) 出来之前, SVM 和 AdaBoost 是效果最好的两个算法,而 AdaBoost 是提升树(Boosting Tree), Boosting 算法是一种把若干个分类器整合为一个分类器的方法,把多个不同的决策树用一种非随机的方式组合起来,表现出惊人的性能! 第一,把决策树的准确率大大提高,可以与 SVM 媲美。第二,速度快,且基本不用调参数。第三,几乎不过拟合。不过要说不会出现过度拟合,那是不正确的,有人说随机森林不会过度拟合,测试后你会发现不一定。

AdaBoost 让决策树起死回生! Breiman 情不自禁地在他的论文里赞扬 AdaBoost 是最好的现货方法( off-the-shelf, 即“拿来就可以用”的意思)。就像 C5.0 和 SVM 那样,我们将训练一个 AdaBoost 分类器来识别 3 种不同的鸢尾花,然后用测试集来测试这个分类器,评估这个模型的准确性,如下所示。

- 数据分组

```
1 require("adabag")
2 train.idx <- sample(1:nrow(iris), 100)
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

只需要写一行 R 代码即可训练 AdaBoost 模型,如下所示。

```
model <- boosting(Species ~ ., data = iris.train)
```

为什么 AdaBoost 训练时间这么长呢? 默认的迭代次数是 100。你可以修改 mfinal 参数来改变迭代的次数; AdaBoost 包含一系列的弱分类器。什么是弱分类器? Boosting 函数使用 AdaBoost.M1 算法,它是一种变异算法,集成了三种弱分类器: FindAttrTest、FindDecRule 和 C4.5。

- 分类测试

```
1 resboost <- predict(object = model, newdata = iris.test, type = "class")
2 resboost$confusion
#               Observed Class
# Predicted Class setosa versicolor virginica
#   setosa         15          0          0
#   versicolor      0         15          1
#   virginica       0          1         18
```

其预测结果中自带了一个混淆矩阵,我们通过检查即可看到模型将 1 个 virginica 误判成了 versicolor,同时将一个 versicolor 误判成了 virginica,即模型认为测试集中有 19 棵 virginica,效果还是不错的。

## 7.8 KNN 算法与 K-means 算法有什么不同

$K$  最近邻，顾名思义，就是  $K$  个最邻近的样本的意思。如果一个样本的最接近的  $K$  个邻居里，绝大多数属于某个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。 $K$  最近邻分类算法是数据挖掘中最简单的方法之一。可以用一句话来形象理解，就是物以类聚，人以群分。

KNN 算法有两个关键点要注意。第一个关键点是  $K$  的确定，选择一个最佳的  $K$  值取决于数据分布情况。总的来说，较小的  $K$  值能使模型更不容易受样本不均衡的影响，而较大的  $K$  值能够减小噪声的影响。第二个关键点是最近邻的定义，也就是距离定义，常用的有欧式距离、余弦距离等，具体采用哪种距离定义要根据实际的数据和业务确定。

由于 KNN 在确定分类决策上只依据最邻近的几个样本的类别来决定待分样本所属的类别，它只与极少量的相邻样本有关，因此它是非线性的，对于类域的交叉或重叠较多的待分样本集来说，KNN 方法非常适用。

其实 KNN 的思想在很多地方都有应用，在网络关系研究中使用得最多，有点像 7.6 节所讲的 PageRank 算法，但 KNN 仅仅描述临近点的个数，而 PageRank 还会考虑临近点的质量。

很多读者容易混淆 KNN 和 K-means，我们这里将其区别整理，如表 7-8 所示，以便大家加以区分理解。

表 7-8

KNN	K-Means
确定观测点应该分到哪一类	目的是将一系列观测点分为 $K$ 群
监督学习分类算法，种类已知	非监督学习聚类算法，将数据关系近的聚在一起从而获得种类
训练数据集有 label 标记类别	训练数据集无 label 标记，算法根据数据关系自行分类标记
此“ $k$ ”是用来计算的相邻数据数。来了一个样本 $x$ ，要给它分类，即求出它的 $y$ ，就从数据集中，在 $x$ 附近找离它最近的 $k$ 个数据点，这个 $k$ 个数据点，类别 $c$ 占的个数最多，就把 $x$ 的 label 设为 $c$	此“ $k$ ”表示类的数目，假设数据集合可以分为 $k$ 个簇，由于是依靠人工设定，需要一点先验知识
$k$ 值确定后每次结果固定	$k$ 值确定后每次结果可能不同，从 $n$ 个数据对象任意选择 $k$ 个对象作为初始聚类中心，随机性对结果影响较大
时间复杂度： $O(n)$	时间复杂度： $O(n \times k \times t)$ ， $t$ 为迭代次数
相似处：给定一个点，在数据集中找离它最近的点，即二者都用到了 NN ( Nears Neighbor )	

我们使用 KNN 算法来识别三种不同类别的鸢尾花，KNN 算法是一种“懒惰”学习算法，它不能训练出一个全局的模型来对数据进行预测，而是对训练集和测试集一起训练和预测，如下所示。

- 数据分组

```
1 library("class")
2 train.idx <- sample(1:nrow(iris), 100)
```

```
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

也就是说需要用 一个训练集来初始化 KNN 分类器，然后用 一个测试集来测试模型性能。

#### • 构建 KNN 模型

```
1 resknn <- knn(train = subset(iris.train, select = -Species), test = subset(
  iris.test, select = -Species), cl = iris.train$Species, k = 2)
2 table(iris.test$Species, resknn)
#           resknn
#           setosa versicolor virginica
# setosa           20           0           0
# versicolor        0          13           1
# virginica          0           1          15
```

上述代码中，训练集的物种列表代表类别；k 用来设置最邻近的个数，最优的 k 值可以通过交叉检验获得。混淆矩阵表示 KNN 较好地完成了任务，仅仅错分了两个。

## 7.9 Naive Bayes（朴素贝叶斯）算法

天下英雄难免观念不同，股票上有技术派和价值派，统计也分两大学派：极大似然学派和贝叶斯学派，因此，贝叶斯是统计计算中一个重要的知识核心。

我们先来看看什么是贝叶斯：在已知类条件概率密度参数表达式和先验概率前提下，利用贝叶斯公式转换成后验概率，最后根据后验概率大小进行决策分类，贝叶斯公式的表达式为

$$P(B|A) = \frac{P(B) \times P(A|B)}{P(A)}$$

然而我们要学习的是朴素贝叶斯，朴素贝叶斯一个重要的假设就是变量独立，换句话说就是各个变量间互不影响， $a$  变量的取值不会影响  $b$  变量取值，因此我们可以把上述公式进一步推导为

$$P(B|A) = \frac{\sum_i^n P(B_i) \times P(A|B_i)}{P(A)}$$

举个例子：

继续使用我们在决策树中的例子，我们将前九个样本的信息作为训练集来训练贝叶斯，把最后一个样本作为测试样本简要测试朴素贝叶斯的效果。

首先计算程序员在各个特征下的概率：

- $P(\text{单身}|\text{程序员}) = 5/6$ ;  $P(\text{有女朋友}|\text{程序员}) = 1/6$ ;
- $P(\text{加班}|\text{程序员}) = 4/6$ ;  $P(\text{不加班}|\text{程序员}) = 1/6$ ;  $P(\text{偶尔加班}|\text{程序员}) = 1/6$ ;
- $P(\text{t-shirt, 拖鞋}|\text{程序员}) = 2/6$ ;  $P(\text{西装革履}|\text{程序员}) = 4/6$ ;

接下来，计算霸道总裁在各个特征下的概率：

- $P(\text{单身}|\text{霸道总裁}) = 1/3$ ;  $P(\text{有女朋友}|\text{霸道总裁}) = 2/3$ ;
- $P(\text{加班}|\text{霸道总裁}) = 1/3$ ;  $P(\text{不加班}|\text{霸道总裁}) = 1/3$ ;  $P(\text{偶尔加班}|\text{霸道总裁}) = 1/3$ ;
- $P(\text{t-shirt, 拖鞋}|\text{霸道总裁}) = 0/3$ ;  $P(\text{西装革履}|\text{霸道总裁}) = 3/3$ ;

有了上面的数据之后，我们就可以预测最后一个样本的职业了：

- $P(\text{程序员}|\text{有女朋友, 不加班, 西装革履}) = 1/6 \times 1/6 \times 4/6 = 1/54$ ;
- $P(\text{霸道总裁}|\text{有女朋友, 不加班, 西装革履}) = 2/3 \times 1/3 \times 3/3 = 2/9$ ;

由于  $P(\text{程序员}|\text{有女朋友, 不加班, 西装革履}) < P(\text{霸道总裁}|\text{有女朋友, 不加班, 西装革履})$ ，所以朴素贝叶斯预测最后一个样本的职业为霸道总裁。

朴素贝叶斯应该是最常见的算法，我们在项目中经常使用这个算法，比如后面讲到的第 10 章等，同样，我们还是使用朴素贝叶斯做一个分类器，识别三种不同的鸢尾花，如下所示。

- 数据分组

```
1 library("e1071")
2 train.idx <- sample(1:nrow(iris), 100)
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

仅需一行代码就可以构建朴素贝叶斯模型，如下所示。

```
modelnb <- naiveBayes(x = subset(iris.train, select=-Species), y = iris.train$Species)
```

调用 `naiveBayes` 函数时，我们把训练集中的物种列去掉之后传递给 `x` 参数，用 `y` 参数指定目标变量。

下面测试一下朴素贝叶斯模型的性能，代码和以前一模一样，如下所示。

```
1 resnb <- predict(object = modelnb, newdata = iris.test, type = "class")
2 table(iris.test$Species, resnb)
#           resnb
#           setosa versicolor virginica
# setosa         17          0          0
# versicolor      0         18          0
# virginica        0          2         13
```

朴素贝叶斯表现得很不错，它只犯了两个错误，把两个 `virginica` 错分成了 `versicolor`。

## 7.10 CART 算法

CART 的英文是 Classification And Regression Tree，直译即为分类回归树算法，简称 CART 算法，它是决策树的一种实现，通常决策树主要有三种实现，分别是 ID3 算法、CART 算法和 C4.5 算法。CART 算法是一种二分递归分割技术，把当前样本划分为两个子样本，使得生成的每个非叶子

节点都有两个分支，因此 CART 算法生成的决策树是结构简洁的二叉树。即使一个 feature 有多个取值，也是把数据分为两部分。如果目标变量是连续的，则 CART 算法找出一组基于树的回归方程来预测目标变量。

其用于选择变量的不纯度度量是 Gini 指数，中文名是基尼系数。如果你是学社科或者经济的，那么肯定知道这个名词，它一开始是用来判断收入分配公平程度的指标，是一种不等性度量，介于 0~1 之间的数，0 即完全相等，1 即完全不相等。在计算收入分配公平程度时，通常需要把收入划分为几个区间，然后再计算基尼系数，这与 CART 的理念是一致的。

Gini 系数的公式如下：

$$\text{Gini}(D_i) = 1 - \sum_i^n P_i^2$$

最终按变量 T 分割的 Gini 系数计算公式为

$$\text{Gini}(D) = \frac{|D_1|}{|D|} \text{Gini}(D_1) + \frac{|D_2|}{|D|} \text{Gini}(D_2)$$

在 CART 算法中主要分为两个步骤：

- (1) 将样本递归划分进行建树过程。
- (2) 用验证数据进行剪枝。

下面举个例子演示一下计算过程，由于有些计算和 C5.0 决策树相似，因此我们只计算加班情况这个变量的基尼系数情况。

由于 CART 每个分支节点都只有两个子节点，而加班情况有三种情况：加班、不加班、偶尔加班，由  $C_3^2=3$  可知，我们要计算三个划分情况，如下所示。

情况一：加班，偶尔加班 VS 不加班

$$\text{Gini}(\text{加班, 偶尔加班}) = 1 - (1/6)^2 - (5/6)^2 = 0.28$$

$$\text{Gini}(\text{不加班}) = 1 - (1/4)^2 - (3/4)^2 = 0.375$$

则加班情况的基尼系数为

$$\text{Gini}(\text{加班情况}) = 1 - 6/10 \times 0.28 + 4/10 \times 0.38 = 0.68$$

情况二：加班，不加班 VS 偶尔加班

$$\text{Gini}(\text{加班情况}) = 1 - 8/10 \times (1 - (4/8)^2 - (4/8)^2) - 2/10 \times (1 - (2/2)^2) = 0.6$$

情况三：不加班，偶尔加班 VS 加班

$$\text{Gini}(\text{加班情况}) = 1 - 6/10 \times (1 - (3/6)^2 - (3/6)^2) - 4/10 \times (1 - (1/4)^2 - (3/4)^2) = 0.55$$

因此，最好的分割方式是情况一。

这里继续使用 CART 算法来识别三个种类的鸢尾花。

- 数据分组

```
1 library("rpart")
2 train.idx <- sample(1:nrow(iris), 100)
3 iris.train <- iris[train.idx, ]
4 iris.test <- iris[-train.idx, ]
```

R 把复杂的 CART 算法封装起来了，因此我们可以用一行代码来建立 CART 分类器。

```
1 modelcart <- rpart(Species ~ ., data = iris.train, method = "class")
```

rpart 函数的 method 需要根据目标变量的类型设定，这里的任务为分类，目标变量为因子变量，所以将方法设定为 class。

- 模型测试

```
1 rescart <- predict(object = modelcart, newdata = iris.test, type = "class")
2 table(iris.test$Species, rescart)
#           rescart
#           setosa versicolor virginica
# setosa           18           0           0
# versicolor        0           15           0
# virginica          0           3          14
```

CART 算法将三个 virginica 错分成了 versicolor，效果并不比上面的监督模型好。

通过十大算法在鸢尾花上的表现，可以比较得出比较肤浅的结论：监督模型在默认的设置下，效果要优秀一些，但是这些还需要细心地调优，并仔细衡量统计意义上的差异。

# 第 8 章

## 数据抓取

### 8.1 数据挖掘工程师不可抱怨“巧妇难为无米之炊”

获取信息的能力往往是成就一个人或一个组织的关键力量，从二战时期的恩格玛密码开始，人类进入了信息时代，信息开始在各个领域发挥越来越大的作用，甚至成为一种独立于其他资源之外的资源。

数据分析挖掘人员除了研究企业自有的数据外，还要能够获得外界的公开数据和二手数据，更要注重内外数据的结合。当自身数据资源匮乏时，更要有能力获得外部数据帮助企业和个人决策。本章我们就集中学习一下 R 抓取数据的技术。

获取信息的能力同其他专业技能一样重要，专业化深度分工的社会里更要注重信息的广度。

下面先介绍怎样用 R 抓取常见的网络数据，抓取数据要虚拟一个命令行浏览器，RCurl 包就是 R 语言的一个命令行浏览器，XML 包则用来解析处理浏览器接受到的 XML 或是 HTML 数据，数据解析完成之后，就要进行一些数据整理的工作，stringr 是处理字符数据的不二之选，如下所示。

- 载入数据包

```
1 if (!suppressWarnings(require(RCurl))) {  
2   install.packages("RCurl")  
3   require(RCurl)  
4 }  
5 if (!suppressWarnings(require(XML))) {  
6   install.packages("XML")  
7   require(XML)  
8 }  
9 if (!suppressWarnings(require(stringr))) {  
10  install.packages("stringr")  
}
```

```
11 require(stringr)
12 }
```

在抓取数据之前，需要了解网络数据的格式，网络数据一般包括 TXT 文本、表格、超文本标记语言（HTML）、JSON 等，另外获取数据时是不是有权限限制等这些需要事先清楚明白，一方面为了选择合适的抓取方法，另一方面如果出现错误或困难，可以详细描述自己的问题方便别人给予帮助。

遵从循序渐进的介绍方法，首先从简单地读取 TXT 格式的网页文本开始，从古腾堡（[www.gutenberg.org](http://www.gutenberg.org)）抓取一本 *An Inquiry into the Nature and Causes of the Wealth of Nations* 读读，这本书的名字直译过来为“国民财富的性质和原因的研究”，它还有一个更为响亮的名字——《富国论》，世人尊称亚当·斯密为“现代经济学之父”和“自由企业的守护神”，人类第一次认识了那双看不见的手（invisible hand），毫不知情的情况下，这双看不见的手引导自私自利的人类在谋取自身利益的同时促进人类全体利益向前发展，今天我们就用 R 完整地抓取这本书，以纪念这一人类精英。

### • 读取文本数据

```
1 url <- "http://www.gutenberg.org/cache/epub/3300/pg3300.txt"
2 fuguolun <- readLines(url)
3 temp <- paste(fuguolun, collapse = "\n")
4 write.table(temp, "H:/zimeiti/探寻数据背后的逻辑：R 语言数据挖掘之道 /
bookwriting/第 8 章数据抓取/data/fuguolun.txt")
```

上述代码中直接使用 R 基础包里的 `readLines` 函数即将《富国论》读取完整了，`readLines` 有很多参数，最有用的是 `n` 和 `encoding` 这两个参数，前者用来指定读取文本的前几行，后者用来指定文本的字符编码。除此之外，`readLines` 函数读取的结果是一个 `list` 对象，文本的每一行（注：文本中一行字符代表着我们所说的一段字符）为 `list` 的一个元素。

如果直接将 `list` 输出为 TXT，R 将在每一段的开头添加 `list` 元素的编号，不符合电子书的格式，需要将它们按一定的格式融合在一起，比如每一行使用换行。使用 `paste` 函数可以将它们粘在一起，需要注意的是，`paste` 函数的两个参数 `sep` 和 `collapse`，它们是用来设置粘贴时以什么分隔符隔离的，用法略有不同。如果要将一个 `vector` 对象粘在一起，使用逗号分开，就设置 `sep = ","`，而不用设置 `collapse`，如果是将一个 `list` 的对象粘在一起，就使用 `collapse = "\n"`。这里是后者，不过选择的分隔符为 `"\n"`。

将调整好的文件使用 `write.table` 函数写入到指定目录，到这里我们就得到了这本名震江湖的《富国论》。

## 8.2 抓取股市龙虎榜数据，碰碰运气

以上是牛刀小试，为自己准备了一份精神食粮《富国论》之后，可以开始尝试抓取一些更有难度的数据。股票应该是很多人学习数据挖掘之路上常常幻想的突破点，不幸的是大多数人在此折戟



沉沙。既然这条路留下了很多“先贤”的足迹，在此也不妨尝试一下抓取股票数据，万一梦想成真了呢？

东方财富网公布大量的股票数据，借此宝地，抓取它们的龙虎榜的股票交易数据留作数据资本也是不错的选择，据我认识的一个位“大隐隐于市”的老股民说，龙虎榜的机构交易数据往往能预示以后的股票走势，机构看好，后期走高，机构逃跑，就有可能成为接盘侠。奉劝各位不要据此投资，以上是不负责任的言语，大可不必当真，成功抓取数据才是本书的责任所在。

## 8.2.1 了解 XML 和 Html 树状结构，才能庖丁解牛

首先解释一下这个网址：<http://data.eastmoney.com/stock/lhb/600006.html> 和 <http://data.eastmoney.com/stock/lhb> 表示东方财富网的数据分支龙虎榜数据；600006 是股票的代码，指东风汽车，可以换成任何已知的股票代码，龙虎榜数据每天收盘后更新，读者可以根据需要抓取；Html 表示网页数据是超文本标记语言格式。

- Html 格式

```
lhb <- "http://data.eastmoney.com/stock/lhb/600006.html"
```

Html 的基础内容是一种描述数据并将数据结构化的语言。在网页上显示数据不仅要标记某一部分是什么文件，比如图片、音频、视频、文本等，还要标注它们的归属关系，Html 和 XML 做的就是这部分工作。在浏览器中鼠标右击选择（快捷键 Ctrl+U）“查看网页源代码”可以查看网页标记语言文本，即头部表示所有超文本语言的几个字符，如下例。

- 简单的 Html 结构

```
1 <html>
2 <body>
3   <h1>烈日灼心</h1>
4   <div id="info">
5     <span><span class='pl'>导演</span>: <span class='attrs'><a href="/
celebrity/1274278/" rel="v:directedBy">曹保平</a></span></span><br/>
6     <span class="actor"><span class='pl'>主演</span>: <span class=
'attrs'><a href="/celebrity/1274235/" rel="v:starring"> 邓 超 </a> / <a
href="/celebrity/1274291/" rel="v:starring">段奕宏</a> / <a href="/celebrity/
1274274/" rel="v:starring">王珞丹</a> / <a href="/celebrity/1118269/" rel="v:
starring">高虎</a></span></span><br/>
7     <span class="pl">制片国家/地区:</span> 中国大陆<br/>
8     <span class="pl">语言:</span> 汉语普通话<br/>
9     <span class="pl">上映日期:</span> <span property="v:
initialReleaseDate" content="2015-08-27(中国大陆)">2015-08-27(中国大陆)</span>
</span><br/>
10  </div>
11 </body>
12 </html>
```

Html 一般分为头部、体部，<html>与</html>之间描述整个网页，包括网页的结构和标志等，<body>与</body>是网页可见的内容，<div>与</div>描述一个模块。其树状结构如图 8-1 所示。

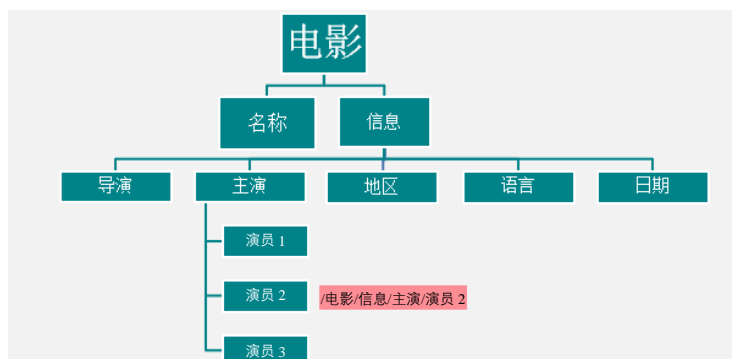


图 8-1

而我们提取网页是从树的根部（root）一层一层向下盘剥的，显示根结构再到枝结构再到亚枝结构再到叶，这样就可以顺藤摸瓜找到目标数据然后抽提出来重新组合成新的数据结构。

### 8.2.2 了解 R Curl 包和网页解析函数

getURL 来自神奇的 R Curl 包，它将网页下载下来，参数很多，一般会用到的只有两个，第 1 个参数为目标网页地址，可以是一个也可以是多个，“encoding”用来指定网页的编码方式，一般为 UTF-8 或 gb2312，若编码参数标注不当则会出现乱码，如下所示。

- 读取 Html 数据

```
1 wp <- getURL(lhb,.encoding = "gb2312")
2 wp2 <- iconv(wp,"gb2312","UTF-8")
3 trade <- htmlParse(wp2, asText = TRUE, encoding = "UTF-8")
```

一旦发现乱码，即为编码方式有误，《密码故事》中写道：“信息接口错误往往会造成信息混乱”。

上述代码中，第 2 行代码用 iconv 函数进行了一次转码，该函数包括 3 个参数，第 1 个为需要转码的数据；第 2 个为数据原始的编码；第 3 个为将要转成的目标编码方式。

要判断抓取数据最重要的部分，就是要对 getURL 函数下载下来的内容进行解析为 R 能识别的 XML/HTML 树状结构。如果网页使用的标记语言不同，那么选择的解析函数也不同，一般可以根据网址的尾部会出现 html 或 xml 来判断，或者在网页源码的开头注明。如果你要解析的是 Html 网页，就需要使用 htmlParse 函数，如果是 XML 网页，就使用 xmlParse 函数，尽管它们之间的参数可以设置相通，但最好还是根据网页源码选择函数。不难看出，龙虎榜是 Html 标记，所以使用 htmlParse 函数解析（见第 3 行代码）。

- 解析函数 (1)

```
1 traderoot <- xmlRoot(trade)
2 temp <- xmlChildren(traderoot)
3 names(temp)
4 tradehead <- xmlChildren(traderoot)$head
5 xmlChildren(tradehead)$title
```

除了解析函数以外，其他整理函数都是通用函数，上述代码中，xmlRoot 函数用于获取网页的顶级节点，顶级节点就是没有父节点的根节点，包括文档自身和子节点。xmlChildren 函数用于查看节点的子节点，函数 names 在 R 中属于基础函数，以后会经常见到，用于查看对象的元素名称，可以查看根节点包括哪些子节点，发现根节点的下一级节点包括“head”和“body”两个子节点。如果使用 xmlChildren(temp) 查看 temp 的子节点就会报错，因为 temp 包含“head”和“body”两个平级的节点，函数不知道要查看的是哪一个的子节点，也就是说，函数只能顺藤摸瓜，不能脚踏两只船。比如可以通过 xmlChildren(tradehead)\$title 查看“head”子节点“title”。

- 解析函数 (2)

```
tradedata <- xpathSApply(traderoot, "//div[@id='container']/div[@class='mainFrame']", xmlValue)
```

当然这样一条一条地查下去非常低效，可以在浏览器中鼠标右击选择（快捷键 Ctrl+Shift+I）“审查元素”查看网页的树状结构。目标数据在“body”分支内，它下面有多个同名的“div”子节点，第 1 个“div”的类（属性）标记为 id='container'，这样就可以指定第 1 个类了，这个类下面又包括多个“div”子节点，但目标数据存储在类（属性）为 class="mainFrame"的“div”子节点内，当在审查元素界面用鼠标指向这个节点时，网页上的表格被阴影覆盖（如图），既然如此就可以通过树状结构指向这个类，如“//div[@id='container']/div[@class='mainFrame']”，两个斜杠表示结构前面省略了一些父分支，xpathSApply 函数用于截取指定树状结构下的分支，如果是表格，table//th 用于指定表格的名称，table//td 指定表格的数据。

## 8.2.3 抓取股票龙虎榜

实际上，针对龙虎榜的数据没必要按照套路抓取，它实际上是两个表格：买入金额最大的前 5 名和卖出金额最大的前 5 名。这样就没必要大费周章的一个节点一个节点地向下抓取。可以通过 readHTMLTable 读取网页中的表格，readHTMLTable 的数据为 htmParse 函数解析的数据，which 参数用于指定读取网页中第几个表格，由于转码后数据的列名出现乱码，这里通过 names 对各列重新命名，如下所示。

- 买入表

```
1 buydf <- readHTMLTable(trade, which = 1)
2 names(buydf) <- c("序号", "交易营业部名称", "买入金额(万)", "占总成交比例", "卖出金额(万)", "占总成交比例", "净额(万)")
```

抓取“卖出金额最大的前 5 名”表格只需要将 readHTMLTable 的 which 参数改为 2 即可，最后

我们需要删除表格 2 的最后一行，因为它仅仅是卖出和买入的汇总值，如下代码所示。这样网页的重点数据已经进入我瓮了，然后就是对数据进行清洗整形。

- 卖出表

```
1 saledf <- readHTMLTable(trade, which = 2)
2 colnames(saledf) <- c("序号", "交易营业部名称", "买入金额(万)", "占总成交比例", "卖出金额(万)", "占总成交比例", "净额(万)")
3 saledf <- saledf[-6,]
```

首先需要对表格进行重新设计以方便存储调取，由于数据每天都会更新，所以需要给数据添加时间列，另外买入卖出分开两个表比较麻烦，不如新建一个交易类型列用于分辨是买入交易还是卖出交易，这样就可以将两个表合并成一个表了。另外，以后也许可能会抓取别的股票的数据，因此添加股票代码列也不可避免，最后形成的表格需要包括股票代码、日期、交易营业部名称、交易类型、交易金额、占总成交比例等 6 列数据。

- 提取公司名称

```
1 tradetitle <- xpathSApply(traderoot, "/html/head/title", xmlValue)
# "东风汽车(600006)龙虎榜数据全览 _ 数据中心 _ 东方财富网"
2 y <- regexpr('\\(.{1,9}\\)', tradetitle, TRUE)
3 z <- y + attr(y, which = "match.length")-1
4 stockcode <- substr(tradetitle, start = y+1, stop = z-1)
```

上述代码中，当 `xpathSApply` 函数的第 3 个参数设置为 `xmlValue` 时，表示提取节点具体的数值，如果不设置表示提取路径下的节点，要提取股票代码就需要提取“/html/head/title”的节点值，而不是节点本身，因为节点是 `XMLInternalElementNode` 类，无法用正则表达式处理。

提取股票代码需要知道代码的起始位置和结束位置，`regexpr` 函数在处理字符串时非常好用，它返回的是模式之前有多少个字符，并且以属性的方式返回模式本身有多长（`match.length`）。比如正则表达式“`\\(.{1,9}\\)`”表示以括号开始且以括号结束，中间包括大概 1 至 9 个字符的模式，例如（600006），它在字符串 `tradetitle` 中的起始位置为 5，模式长度为 8。`regexpr` 函数的第 1 个参数用于指定正则表达式模式，第 2 个参数指定数据，第 3 个参数用于设置时兼容 `perl` 模式。

根据 `regexpr` 函数结果，股票代码的起始位置已经获得，结束位置可以通过起始位置加上模式长度进行计算。`attr` 用于提取对象的属性（`attributes`），它有两个参数，`which` 参数用于指定属性名称。`substr` 用于提取字符串中指定位置的字符，`start` 和 `stop` 参数用于指定字符起始位置和结束位置，这样就提取到了股票的代码。其实这一段完全没有必要这么做，因为在抓取网页时肯定已经存储了股票的代码，但是为了讲解字符串的处理此处先行叙述一下。

- 整理买入表

```
1 buydf <- buydf[, 2:4]
2 stockcode1 <- rep(stockcode, length(buydf[, 1]))
3 buydf <- cbind(stockcode1, buydf)
4 date <- rep(Sys.Date(), length(buydf[, 1]))
5 buydf <- cbind(buydf, date)
6 changetype <- rep("买入", length(buydf[, 1]))
```

```

7 buydf <- cbind(buydf, changetype)
8 colnames(buydf) <- c("股票代码", "交易营业部名称", "交易金额", "占总成交比例",
"日期", "交易类型")

```

上述代码是为数据框 buydf 添加股票代码列，第 1 行代码提取有数据的 2~4 列，组成新的数据框；第 2 行代码使用 rep 函数将股票代码重复与 buydf 行数相同的次数；第 3 行代码使用 cbind 函数将两个对象添加在一起；第 4 行代码为数据添加日期列，假设当天的龙虎榜数据当天抓取，那么可以使用 Sys.Date 获得系统日期（必须保证系统时间不是错误）；第 5~8 行代码添加交易类型列，由于第一个表全是买入机构，交易类型全部标记为买入，完成需要添加的列后，原有的列名就需要改变一下了，可以使用 colnames 或 names 函数。

与处理 buydf 的过程相似，处理 saledf 数据框的程序没有太大变化，首先目标数变为原表的第 2、5、6 列，先行提取出来，然后依次添加股票代码、日期、交易类型（卖出）3 列，再对数据列重新命名，如下所示。

- 整理卖出表

```

1 saledf <- saledf[, c(2, 5, 6)]
2 stockcode1 <- rep(stockcode, length(saledf[, 1]))
3 saledf <- cbind(stockcode1, saledf)
4 date <- rep(Sys.Date(), length(saledf[, 1]))
5 saledf <- cbind(saledf, date)
6 changetype <- rep("卖出", length(saledf[, 1]))
7 saledf <- cbind(saledf, changetype)
8 colnames(saledf) <- c("股票代码", "交易营业部名称", "交易金额", "占总成交比例",
"日期", "交易类型")

```

下面将两个表合成一个表，抓取的数据把数值型变量当成因子，因此要重新转化为数值型变量，因子转化为数值时首先要转化为向量，再转化为字符。

- 合并表

```

1 lhbdata <- rbind(buydf, saledf)
2 lhbdata$交易金额 <- as.numeric(as.vector(lhbdata$交易金额))
3 capitalin <- sum(lhbdata[which(lhbdata$交易类型 == "买入"), 3]) -
sum(lhbdata[which(lhbdata$交易类型 == "卖出"), 3])

```

这样我们就可以计算前 10 名机构的资金是净流入还是净流出了，结果很令人失望，东风汽车的资金在净流出。

## 8.2.4 资金流入分析

如果股票有资金持续流入，就是一直需要关注的股票，很多数据分析师都是从分析证券之间的联系开始的，可能他们都有一个靠分析赚钱的美好梦想，后来这个梦想也大都实现了，不过多是放弃股市后依靠分析工作的薪水来积累财富的。

下面对资金注入进行简单分析，代码如下所示。

- 载入数据包

```
1 if (!suppressWarnings(require("ggplot2"))) {
2   install.packages("ggplot2")
3   require("ggplot2")
4 }
5 if (!suppressWarnings(require("reshape2"))) {
6   install.packages("reshape2")
7   require("reshape2")
8 }
9 if (!suppressWarnings(require("plyr"))) {
10  install.packages("plyr")
11  require("plyr")
12 }
13 if (!suppressWarnings(require("grid"))) {
14  install.packages("grid")
15  require("grid")
16 }
17 if (!suppressWarnings(require("gridExtra"))) {
18  install.packages("gridExtra")
19  require("gridExtra")
20 }
21 if (!suppressWarnings(require("extrafont"))){
22  install.packages("extrafont")
23  require("extrafont")
24 }
```

- 载入自备绘图函数

```
1 add_credits = function(fontsize = 12) {
2   grid.text("大音如霜工作室",
3           x = 0.99,
4           y = 0.02,
5           just = "right",
6           gp = gpar(fontsize = fontsize, col = "#777777", fontfamily =
'STXingkai'))
7 }
8 title_with_subtitle = function(title, subtitle = "") {
9   ggtitle(bquote(atop(. (title), atop(. (subtitle)))))
10 }
11 #font_import()#引入字体
12 loadfonts(device="win")
```

- 数据整理

```
1 temp <- lhldata[which(lhldata$交易类型 == "卖出"),]
2 temp <- temp[order(temp$交易金额),]
3 temp$交易金额 <- round(temp$交易金额)
```

```
4 temp$交易营业部名称 <- factor(temp$交易营业部名称, levels = temp$交易营业部名称)
```

上面的函数大多数已经在第 3 章中介绍过了, 这里不再赘述。简单说一下数据整理部分, 首先选出做空东风汽车的机构, 然后使用 `order` 函数对交易金额进行升序排序, `order` 返回的是该数值在原序列的位置编号, 比如升序排在第 1 位的最小值在原序列中位于位置编号为 5。使用 `round` 函数将交易金额保留到整数位。由于绘制条形图时, `ggplot` 会按照因子的 `levels` 顺序对条形排序 (如果是字符则按照字符顺序排序), 而不是按照数值大小, 这样绘制出来的图就比较凌乱, 因此需要使用 `factor` 函数重新定义“交易营业部名称”这个因子的 `levels` 顺序, 按照排序后的因子出场顺序定义。

- 绘制卖出机构前 5 名的条形图

```
1 p <- ggplot(temp,aes(x = 交易营业部名称, y = 交易金额)) +
2   geom_bar(stat = "identity", fill = rgb(red = 0, green = 130, blue = 137,
max = 255)) +
3   coord_flip() +
4   geom_rect(aes(xmin = 0.5, xmax= 5.5, ymin= 5300, ymax=5800),
5             fill = rgb(red = 239, green = 153, blue = 165, max = 255)) +
6   geom_text(aes(x = 1:5, y = rep(5550,5)), label = temp$交易金额,
7             colour= "white",
8             family = 'Microsoft YaHei') +
9   title_with_subtitle("做空东风汽车机构前 5 名") +
10  ylab("") +
11  xlab("") +
12  theme_bw(18) +
13  theme(plot.margin = unit(c(1, 1, 1.25, 0.5), "lines"),
14        panel.background= element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
15        plot.background= element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
16        plot.title = element_text(size = rel(1.2), family = 'STXingkai' ,
17                                  face = 'bold', hjust = -0.05,
18                                  vjust = 1.5, colour = '#3B3B3B'),
19        panel.grid.major = element_line(colour=rgb(red = 146, green = 146,
blue = 146, max = 255),size=.75),
20        panel.border = element_rect(colour=rgb(red = 242, green = 242, blue
= 242, max = 255)),
21        axis.ticks = element_blank(),
22        axis.text.x = element_blank(),
23        axis.text.y = element_text(colour="grey20",size=12),
24        axis.title.y = element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=.5),
25        axis.title.x=element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=-.5),
26        legend.position="")
27 png(filename = "H:/zimeiti/plot/sales.png", width = 800, height = 400)
```

```
28 print(p)
29 add_credits(fontsize = 14)
30 dev.off()
```

效果如图 8-2 所示。ggplot 的相关函数已经在第 3 章进行了非常详细的讲述，绘制图 8-2 所用的函数几乎都可以从第 3 章找到，其中 theme 函数几乎是固定的风格。

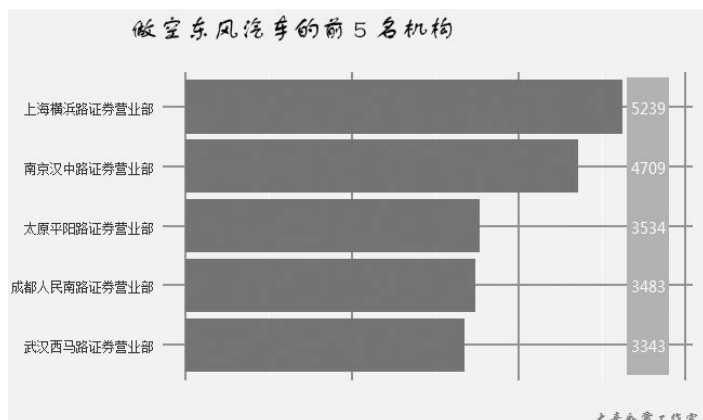


图 8-2

其实股票市场岂是数据挖掘能够战胜的，机构除了比散户拥有资金优势以外，还具有明显的信息优势，它们除了能够拿到上市公司的最新信息，还可以利用资金优势唆使上市公司配合它们发布信息，另外散户的一举一动，券商的数据库里都有记录，信息差距如此巨大可见一斑。所以与其沉迷于复杂的模型中不能自拔，倒不如通过跟踪机构，选出具有指标意义的机构对象，跟着它们买卖也不失为良策。

另外本书的电子版代码中还分享了抓取所有龙虎榜的数据代码，以及几篇翻译的关于股票分析的文章，感兴趣的读者可以去下载。

## 8.3 抓取某家医药信息网站全站药品销售数据

今天我们要学习一下抓取上市医药公司的报表数据，某家医药信息网站公布了欧美大型上市医药企业大量的药品数据，这里就尝试将该网站所有的药品数据抓取下来，以供本书相关分析章节使用。

### 8.3.1 所有医药公司名称一网打尽

我们依然采取顺藤摸瓜的策略，抓取这个数据分为三步：第一步抓取医药公司名称，制作成表格备用；第二步根据公司名称查询公司名下的所有药品名称；第三步根据药品名称查询药品历年的



销售明细及其他相关说明信息。

下面开始进行第一步，抓取医药公司名称。

- 加载包

```
1 if (!suppressWarnings(require(RCurl))) {
2   install.packages("RCurl")
3   require(RCurl)
4 }
5 if (!suppressWarnings(require(XML))) {
6   install.packages("XML")
7   require(XML)
8 }
9 if (!suppressWarnings(require(stringr))) {
10  install.packages("stringr")
11  require(stringr)
12 }
13 if (!suppressWarnings(require("reshape2"))) {
14  install.packages("reshape2")
15  require("reshape2")
16 }
17 if (!suppressWarnings(require("plyr"))) {
18  install.packages("plyr")
19  require("plyr")
20 }
```

## 1. 使用老套路解析网页

- 下载解析网页

```
1 company <- "http://consensus.druganalyst.com/AstraZeneca/Drugs/"
2 company <- getURL(company, .encoding="UTF-8")
3 company <- htmlParse(company, asText = TRUE, encoding = "UTF-8", useInternalNodes=
TRUE )
4 comroot <- xmlRoot(company)
5 company <- xpathSApply(comroot, "//div[@id='menuContainer']/div[@id=
'menuCompanies']/div/ul/li/a")
```

上述代码的前 4 行就不再赘述了，首先下载网页，然后使用 `htmlParse` 函数进行解析，再使用 `xmlRoot` 函数提取根节点。第 5 行代码使用浏览器的“审查元素”功能，找到公司名称在路径“`div[@id='menuContainer']/div[@id='menuCompanies']/div/ul/li/a`”下，因此使用 `xpathSApply` 函数提取所有该路径下的节点，到这里目标内容已经全部存储在 `company` 里了。

## 2. do.call (“土豪”)的生活你不理解

在处理数据的过程中，我们会经常碰到一些后续工作中不会用到的中间结果，这些变量会占用大量的内存空间，这里首先给大家介绍一个笨方法（简称技巧），可以将这些变量统一命名为 `temp1`

和 temp2，只要碰到这类变量就以固定的名字命名，这样上一步的对象 temp1 值就会被新的赋值替换掉，不会在内存里保存大量的无用对象。呵，确实是笨方法，真是难以启齿啊。好吧，之所以使用这种方法是因为我已经懒惰到翻一翻眼皮都觉得费时的地步，如果你想高大上一点，也可以使用 Hadley Wickham 的 dplyr 包，该包将数据当作数据流向下流淌，中间变量不再保存在内存中。至于 Hadley 本人，他已经被人捧为革命者，高大上到我都不好意思简单介绍的程度了，用他自己的话说，即“如今名誉已经到达了令人不安的水平”。

- 结构化输出结果

```
1 temp1 <- lapply(company, xmlAttrs)
2 temp1 <- data.frame(do.call("rbind", temp1))
3 region <- c(rep("European Pharma", 20), rep("US Pharma", 35), rep("Japanese Pharma", 9))
4 temp1 <- cbind(region, temp1)
5 companyname <- unlist(lapply(company, xmlValue))
6 company <- cbind(temp1, companyname)
7 company <- data.frame(lapply(company, as.character), stringsAsFactors = F)
8 write.csv(company, "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取/data/company.csv", row.names = FALSE)
```

在超文本标记语言中分支可能具有一定的属性，如<div id="menuContainer" class="dd-menu">表明 div 分支有两个属性 id 和 class。xmlAttrs 函数用于提取分支的属性，其输出结果是一个 list 对象。

do.call 函数是一个非常有用而又一言半语难以说清的函数，首先这个函数是一个调用函数，请注意调用分为两部分，即 call 和 do 部分，不仅调还要用。另外 do.call 函数将函数名称转化为字符向量，比如你要调用 ggplot 函数，直接输入名称就可以了，但是如果需要一次调用多个函数，就需要将函数名转化为字符向量(该向量包含所有你要调用的函数名)，而 do.call 函数实现了这一步。另外，当函数的参数无限多时，你就需要一个 list 来存储这些参数，比如上面的 rbind 函数的参数就可以无限增长，比如要将三个等长向量绑定在一起，你需要输入 rbind(v1, v2, v3)，如果是将一个 list 里的 10000 个向量捆绑在一起，难道你要将其写为 rbind(v[1], v[2], .....v[10000])，够了，还是请出 do.call 吧，它的第 1 个参数为需要调取的函数名称，第 2 个参数是一个 list，包含被调用函数所需的所有参数。这里调用了 rbind 函数将分支的属性按照行粘贴在一起，然后通过 data.frame 函数转化为数据框。

我们一共抓取了 64 家医药公司，前 20 家是欧洲企业，中间 35 家是美国企业，最后 9 家是日本企业。首先通过 rep 函数将企业归属地重复相应的次数，然后使用 cbind 函数将归属地 and 公司名称捆绑在一起，将整个数据框的变量类型全部转化为字符型，最后输出保存结果。

### 8.3.2 为什么抓取数据时可以使用 For 循环

第二步是根据公司名称寻找公司药品，创建一个空数据框用来存储数据，使用数据框 company 中的 href 变量赋值给一个临时变量 temp，用来构建网页地址。形如 <http://consensus.druganalyst.com/AbbVie/Drugs> 的网页地址可以分为三个部分：前半部分为 <http://consensus.druganalyst.com>，指明

了网站地址和用户类型；/AbbVie 为公司名，存放在向量 temp 里（包括）；/Drugs 是个固定的尾坠。

- 按公司名索取药品名

```

1 companydrug <- data.frame()
2 temp <- company[, "href"]
3 for (i in 1:length(temp)) {
4   s <- sample(1:5, 1)
5   Sys.sleep(s)
6   url <- paste("http://consensus.druganalyst.com", temp[i], "/Drugs", sep
= " ")
7   drug <- getURL(url, .encoding="UTF-8")
8   drug <- htmlParse(drug, asText = TRUE, encoding = "UTF-8", useInternalNodes
= TRUE )
9   table1 <- readHTMLTable(drug, which = 1)
10  if (ncol(table1) > 1)
11    table1 <- as.vector(table1[, -2])
12  table1 <- as.data.frame(table1)
13  table1 <- table1[-(which(table1[,1] == "R&D PIPELINE"):length (table1
[,1])),,]
14  table1 <- as.vector(table1)
15  x <- rep(temp[i], length(table1))
16  table1 <- cbind(x, table1)
17  companydrug<- rbind(companydrug, table1)
18 }
19 colnames(companydrug) <- c("href", "brandname")
20 companydrug <- join(company, companydrug)

```

你来查询网页，别人提供网页供你查询，在一定程度上双方是和谐的。但现在打破规则抓取数据，别人就要防备数据被克隆，超过了度，双方就变成了攻防战。

然后就是老一套了，paste 函数将网址组成的片段装成完整的地址，getURL 函数下载网页，htmlParse 解析网页。药品名称就存储在网页的第 1 个表格中，使用函数 readHTMLTable 提取表格，因为只需要提取药品名称，所以将第 2 列删除仅保留第 1 列，注意在这里做了一次判断，因为有些网页表格有两列，有些只有一列，然后将表格转化为数据框。除了删除无用的列外，还要将无用的行删除，表格从包含 R&D PIPELINE 字样的这行以下都是无用的行，予以删除。将提取的药品名由数据框转化为向量，将公司名复制和药品数相同的次数，并与药品名按列 cbind 合并在一起，使用 rbind 函数存储于 companydrug 数据框，这样就初步完成了药品名的抓取。

抓取药品名称初步完成之后就要进行一次数据整理，以方便下一步抓取药品明细使用。首先对数据框 companydrug 的两列数据命名，第 1 列命名为 href，与 company 数据框里保持一致，因为使用 join 函数将两个数据框合并时，join 函数将两个数据框中名称相同的列作为 id 列进行数据框合并了。

### 8.3.3 不要把代码写复杂

下面即将抓取药品明细，想到即将把一个网站的数据几乎全部抓取下来，心里还有点小激动。另外 druganalyst 这个网站还是比较出类拔萃的，踏踏实实地做了些数据行业的分内之事，比如简单预测药品未来几年的销量，还添加了医药行业对药品的市场评价等一些落地实在的信息功能。

数据明细的页面地址为 `http://consensus.druganalyst.com/Guest/AbbVie/Humira`，同样可以根据已经抓取的配件分为三个部分：`http://consensus.druganalyst.com/Guest` 为固定的头部；`/AbbVie` 存储在已经抓取的 href 变量里；`/Humira` 就是反斜杠加上药品名（brandname）。

但是 brandname 里面的变量名还不能完全用来构建页面地址，例如 `http://consensus.druganalyst.com/Guest/Bayer/Yasmin_Yaz`，对应的药品名 brandname 为 `Yasmin/Yaz`，显而易见，需要清洗处理一下才能作为构建 URL 的材料。如下所示。

- 数据清洗

```
1 urlbrand <- companydrug[, "brandname"]
2 companydrug <- cbind(companydrug, urlbrand)
3 companydrug[] <- lapply(companydrug, as.character)
4 companydrug <- companydrug[grepl("\\S", companydrug$brandname), ]
5 companydrug[, "urlbrand"] <- gsub("/", "__", companydrug[, "urlbrand"])
6 companydrug[, "urlbrand"] <- gsub(" ", "%20", companydrug[, "urlbrand"])
7 companydrug[, "urlbrand"] <- gsub("-", "_", companydrug[, "urlbrand"])
8 companydrug[236, "urlbrand"] <- "Polio__Whooping__Hib%20Vaccine"
9 companydrug[238, "urlbrand"] <- "Meningitis__Pneumonia%20Vaccine"
10 companydrug[360, "urlbrand"] <- "talimogene%20aherparepvec"
11 companydrug[380, "urlbrand"] <- "Reveglucosidase%20alfa"
12 companydrug[554, "urlbrand"] <- "MK_5172%20__plus_%20MK_8742"
13 write.csv(companydrug, "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取/data/companydrug.csv", row.names = FALSE)
```

看到了吧，末了还生出一大堆事端。抓取网页要求首先要对网页非常熟悉，否则工作就会教你在流程中慢慢熟悉。从抓取网页这件事上，慢慢认识到将任务恰当地分解也是现代职场中至关重要的能力，这种能力来源于现代社会专业化细致分工的需要。比如有些人看到一张图表，除了觉得美丑以外就是觉得自己做不出来，而有些人就可以迅速将这张图表分解成数轴、背景、绘图区、图表区、刻度、图例等细节，然后根据各个细节的特点重新组合为整图，那么即使这幅图不能马上做出来，心里也已经有个子丑寅卯了。

说到数据清理，首先要复制 brandname 成为一个新变量 urlbrand，专门用于构建网页地址 URL，然后将 urlbrand 并入数据框 companydrug，再将数据框的变量由因子类型全部转化为字符型变量，这一句代码虽然短小却让人感到丈二和尚摸不着头脑，这句后面的 `lapply(companydrug, as.character)` 产生一个 list，但经过这一句赋值就变为数据框，重点在 `companydrug[]` 里，这个“[]”符号具有将 list 转化为数据框然后再赋值的功能，如果你去掉这个符号，那么 `companydrug` 就不再是数据框了，而是 list，因此这一句等同于“`companydrug <- data.frame(lapply(companydrug, as.character))`”，

stringsAsFactors = F)，个人建议还是这种写法清晰明了，上面的写法虽然也可以但这正是 R 代码丑陋不堪为人诟病的原因，这里特别指出来以警示大家不要过分追求新奇，代码除了实现功能以外还要易于交流沟通，可读性好才是真的好。

整理好数据框后，需要整理一下 urlbrand 变量，例如将 Yasmin/Yaz 中的反斜杠替换为下画线“\_”，检查 urlbrand 变量发现还需要将空格替换为“%20”、将“-”替换为“\_”等，函数 gsub 即为使用正则表达式执行替换的函数，它有 3 个变量，第①个为被替换正则表达式模式（可以是向量，以实现批量替换），第②个为将要替换成的正则表达式模式，第③个为数据集。除此之外我们还发现了一些需要特别处理的数据点，例如“Polio/Whooping/Hib Vac…”需要替换为“Polio\_\_Whooping\_\_Hib%20Vaccine”等，如果在执行下面抓取药品明细数据的循环时报了错，就需要看看循环停在了哪里，然后查看对应的 URL 地址是否有问题，如果有问题更改一下就可以延续停止的地方继续循环执行了。最后将清洗过的数据框写入磁盘备用。

1. 学会按数据点设定规则

下面的代码看起来很复杂，其实大部分内容在提取公司名和药品名称的过程中已经见识过了，几乎没有任何新的内容，之所以看起来那么复杂，是因为提取的页面内容确实比较多，图 8-4 里的所有数据点几乎都有涉及。

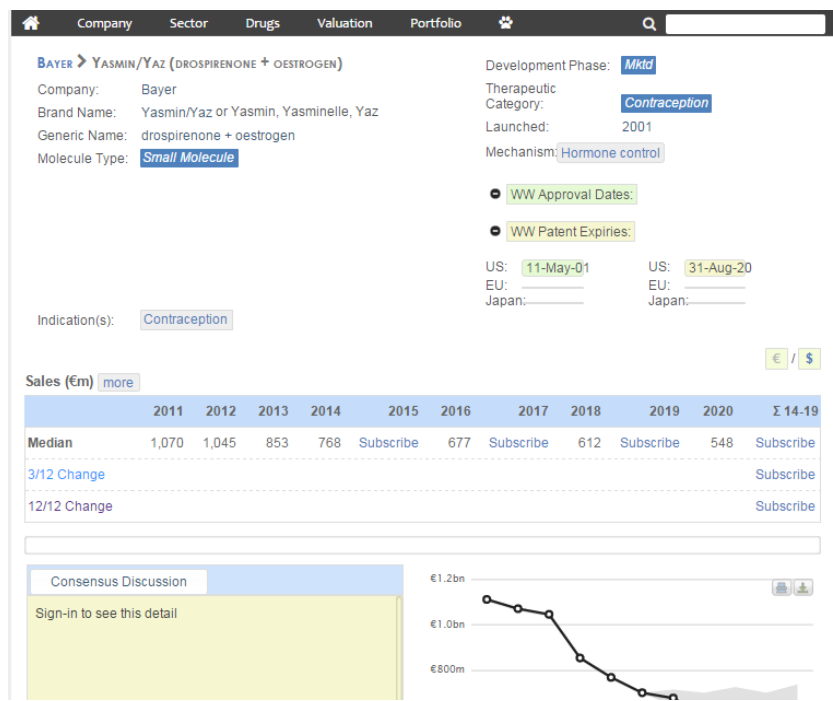


图 8-4

## 探寻数据背后的逻辑：R 语言数据挖掘之道

第三步，按公司名\药品名索取销售明细。

```
1 details <- data.frame()
2 for (i in 1:length(companydrug[, "urlbrand"])) {
3   s <- sample(1:5, 1)
4   Sys.sleep(s)
5   url <- paste("http://consensus.druganalyst.com/Guest", companydrug[i, "
href"], "/", companydrug[i, "urlbrand"], sep = "")
6   wp <- getURL(url, .encoding = "UTF-8")
7   drug <- htmlParse(wp, asText = TRUE, encoding = "UTF-8", useInternalNodes
= TRUE )
8   table1 <- readHTMLTable(drug, which = 1)
9   names(table1)[1] <- "R"
10  table1 <- table1[-c(2,3), ] #删除空白行
11  table1 <- table1[, -ncol(table1)] #删除最后求和列
12  table1 <- data.frame(lapply(table1, as.character), stringsAsFactors = F)
13  table1 <- melt(table1, id.var = 1) #以第一列为 id 列将列有 wide 型变为 long 型
14  comroot <- xmlRoot(drug)
15  companybrandgeneric <- xpathSApply(comroot, "//div[@class='drug-
section-1']/div/
16      span[@class='dtxx']", xmlValue)
17  moleculetype <- xpathSApply(comroot, "//div[@class='drug-section-1']
/div/
18      span[@class='info-lbl']", xmlValue)
19  developeptarget <- xpathSApply(comroot, "//div[@class='drug-section-2']
/div/
20      span[@class='info-lbl']", xmlValue)
21  launchdate <- xpathSApply(comroot, "//div[@class='drug-section-2']/div/
22      span[@class='dtxx']", xmlValue)
23  mechanism <- xpathSApply(comroot, "//div[@class='drug-section-2']/div/
24      span[@class='dtxx notice-lbl']", xmlValue)
25  approvaldate <- xpathSApply(comroot, "//div/span[@class='dtxx
green-lbl- 2']", xmlValue)
26  patentexpiric <- xpathSApply(comroot, "//div/span[@class='dtxx yellow-
lbl-2']", xmlValue)
27  moneytype <- xpathSApply(comroot, "//div[@class='companyDrugTable_
Header ']", xmlValue)
28  moneytype <- gsub("^.{10}", "", moneytype) #去除前后不必要的字符
29  moneytype <- gsub(".{3}$", "", moneytype)
30  review <- xpathSApply(comroot, "//div[@id='daview']", xmlValue)
31  review <- gsub("\\r", "", review)
32  review <- gsub("\\n", "", review)
33  review <- gsub("\\t", "", review)
34  companyname <- rep(companybrandgeneric[1], length(table1[,1]))
35  brandname <- rep(companybrandgeneric[2], length(table1[,1]))
```

```

36 genericname <- rep(companybrandgeneric[3], length(table1[,1]))
37 moleculetype <- rep(moleculetype, length(table1[,1]))
38 developmentphase <- rep(developoptarget[1], length(table1[,1]))
39 therapeuticcategory <- rep(developoptarget, length(table1[,1]))
40 launchdate <- rep(launchdate, length(table1[,1]))
41 mechanism <- rep(mechanism, length(table1[,1]))
42 approvaldateus <- rep(approvaldate[1], length(table1[,1]))
43 approvaldateeu <- rep(approvaldate[2], length(table1[,1]))
44 approvaldatejapan <- rep(approvaldate[3], length(table1[,1]))
45 patentexpirieu <- rep(patentexpirieu[1], length(table1[,1]))
46 patentexpirieu <- rep(patentexpirieu[2], length(table1[,1]))
47 patentexpiriejapan <- rep(patentexpirieu[3], length(table1[,1]))
48 moneytype <- rep(moneytype, length(table1[,1]))
49 review <- rep(review, length(table1[,1]))
50 href <- rep(companydrug[i,"href"], length(table1[,1]))
51 table1 <- cbind(companyname, brandname, genericname, moleculetype,
developmentphase, therapeuticcategory, launchdate, mechanism, approvaldateus,
approvaldateeu, approvaldatejapan, patentexpirieu, patentexpirieu,
patentexpiriejapan, moneytype, review, href, table1)
52 details <- rbind(details, table1)
53 }
54 details <- unique(details)
55 write.csv(details, "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取
/data/details.csv", row.names = FALSE)

```

下面简单介绍一下提取过程：首先创建了一个空数据框 details，用来存储数据，循环开始时随机暂停几秒以免 IP 被封，构建一个完整的药品信息页面地址 URL，下载网页之后解析网页。将最重要的销售数据表格解析出来，因为它是页面第一个也是唯一的一个表格，所以使用 readHTMLTable 函数提取。表格的第 1 列没有名称，所以使用 name 函数随意命名为 R。表格后两行是无用的空白数据，所以将其删除，同时删除表格最后的汇总列。然后将表格变量由因子型转化为字符型变量。最后将表格由 wide 型转化为 long 型，这样表格部分就完成提取整理了。

下面需要提取其他相关的药品属性，首先使用 xmlRoot 函数提取网页的根节点，然后根据相应数据所在的节点和节点属性提取即可，比如药品的品牌名（商品名）和通用名所在的节点同级同属“//div[@class='drug-section-1']/div/span[@class='dtxi']”，即可一并提取，下面的提取过程大同小异，不再赘述，值得一提的是，药品这种特殊的商品存在两个名字，品牌名也就是商品名，不同厂家的商品名可能不同，但是通用名就必须相同，比如白加黑是其商品名（拜耳生产），施贵宝生产的商品名就叫日夜百服宁，而其通用名皆为氨酚伪麻美芬。

## 2. 批量抓取的复杂与简单

上面一一提取节点的过程看起来多有重复，比较复杂，作为初学者阅读起来比较易于理解，但是作为一个稍有追求的常年混迹“爬虫”界的人士，任何重复的东西都是不可接受的，我们不妨将

其改造一下，需要说明的是如果你的简化不能提高代码的性能，那么行数变得再少也没有意义。请看如下代码。

```
1 node <- c("//div[@class='drug-section-1']/div/  
2         span[@class='dtxt']", "//div[@class='drug-section-1']/div/  
3         span[@class='info-lbl']", "//div[@class='drug-section-2']/div/  
4         span[@class='info-lbl']", "//div[@class='drug-section-2']/div/  
5         span[@class='dtxt']", "//div[@class='drug-section-2']/div/  
6         span[@class='dtxt notice-lbl']", "//div/span[@class='dtxt  
green-lbl-2']", "//div/span[@class='dtxt yellow-lbl-2']", "//div[@class='  
companyDrugTable_Header']", "//div[@id='daview']")  
7 xpathSApply(doc = comroot, path = node, xmlValue))
```

这样看起来比原来的代码简单多了，这里就把所有将要提取的节点按其属性存储为一个向量，使用 `xpathSApply` 函数批量提取，`xpathSApply` 函数属于 `lapply` 家族的衍生品种。除了代码变得简洁而且速度也可以相应提高，但是存在后遗症，这样产生的结果为向量，而向量的元素很难归类整理，即使按顺序整理好了也难免因为某个网页缺少元素而引起信息整理错位，所以如果不是大量抓取还是要慎用，否则谁头疼谁知道。

节点提取完成之后，需要对数据进行清洗整合，首先在货币类型（`moneytype`）中提取到的内容是这个样子的“`\r\n\tSales (¥bn)\r\n`”，这些符号是什么，必须删除。通过 `gsub` 函数将其前 10 个字符和后 3 个字符替换掉，“`^`”和“`$`”在正则表达式中分别代表从头匹配和从未匹配的意思。然后就剩下了整洁的货币单位了，整理一下 `review` 变量。整理完成之后依次将每一个数据点复制了与 `table1` 长度相同的次数，再将它们和 `table1` 按列捆绑在一起就可以了。最后将 `table1` 和 `details` 按行捆绑在一起就完成了数据的初步提取，写入硬盘（不输出行序号）保存备用，别忘了使用 `unique` 函数去除重复。

### 8.3.4 用 Sankey 数据流描绘医药市场份额流动

到这里，某家医药信息网站的数据抓取任务就基本完成了，希望这个过程能够帮助大家学到一些知识，重要的是学会不惧事艰，分解任务，按步骤一步一步完成。下面简单地做一次报表分析，最近基因蛋白质等大分子药物市场增长强劲，首先看看它们和传统小分子药物的市场比例，然后再分析一下各公司所占的市场份额，最后我们还想知道欧洲、美国、日本 2014 年谁能称雄，其实上面的所有问题都可以转化为份额比较完成，直接制作饼图就可以了，但是这里将饼图变形一下，在一张图中完成。

#### 1. 绘制全球医药行业 Sankey 图

- 载入数据包

```
1 require(igraph)  
2 require(rCharts)  
3 require(rjson)
```



```
4 require(plyr)
5 require(reshape2)
#https://github.com/timelyportfolio/rCharts_d3_sankey 下载文件
```

上面有些包不在 R 语言官方源中, 需要从 Github 中下载安装, 安装方法在第 1 章中将已详细讲解, 当然网络上也可以找到安装方法, 需要强调的是那个注释的地址, 将相关的文件下载下来并解压到本地, 因为其文件路径将用来设置下面图表的 setLib 参数, 以调用相关的库。

读取数据之后需要清洗整理数据, 首先数据中的货币单位(请注意是货币种类和单位)不统一, 所以我们将其统一转化为百万美元(\$m), 数据中一共出现了 6 种货币单位, 为了减少篇幅和困难, 仅按照近期货币兑美元汇率转换。使用 join 函数给 details 匹配上汇率, 同时匹配上企业所在地(region 变量在 company 表), 然后筛选出 2014 年的销售数据, 并提取报表所需要的列, 赋值给新的数据框 data, 如下所示。

#### • 数据准备 (1)

```
1 details <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取
/data/details.csv", header = T, sep = ",", stringsAsFactors = F)
2 company <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取
/data/company.csv", header = T, sep = ",", stringsAsFactors = F)
#details <- details[, -1]
3 capital <- data.frame(moneytype = c("CHFm", "$m", "€m", "£m", "DKKm", "¥bn"),
exchange = c(1.0493, 1, 1.1345, 1.5439, 0.1521, 8.4))
4 details <- join(details, capital)
5 details <- join(details, company[, c("region", "href")])
6 data <- details[which(details$variable == 2014),]
7 data <- data[, c("region", "companyname", "moleculetype", "value",
"exchange ")]
```

value 变量有很多空白值而且是字符型变量, 这类空白值并不是 NA 也不是 NULL, 这里采取一种迂回的方式删除了它们, 即通过正则表达式筛选出非空白字符。另外 value 是以千分计数, 数字间存在 “,” 导致其格式变为字符型, 所以先将逗号替换掉, 然后转化为数值型变量。

#### • 数据准备 (2)

```
1 data <- data[grepl("\\S", data$value), ]
2 data$value <- gsub(",", "", data$value)
3 data$value <- as.numeric(data$value)
4 data <- transform(data, sales = round(value * exchange))
5 temp <- aggregate(sales ~ companyname, data, sum)
6 temp <- temp[order(temp$sales, decreasing = TRUE),]
7 temp <- temp[1:30,]
8 names(temp) <- c("companyname", "total")
9 data <- join(temp, data)
```

上述代码中, transform 函数用于在数据框中通过计算添加新的列, 这里添加了汇率转化后的 sales 列。因为公司太多, 仅选取销售额排名前 30 的公司, 所以需要对各公司总销售额汇总, 使用透视函数 aggregate, 记住, 这个函数等于 Excel 透视表的功能, 然后降序选择前 30, 并给 temp 的列命名。

join 函数在默认情况下进行的左匹配，等于筛选出了销售额前 30 的销售明细。

- 数据准备 (3)

```
1 temp1 <- data[, c("moleculetype", "companyname", "sales")]
2 names(temp1) <- c("source", "target", "value")
3 temp2 <- data[, c("companyname", "region", "sales")]
4 names(temp2) <- c("source", "target", "value")
5 temp <- rbind(temp1,temp2)
```

要模仿一幅图，首先要知道对方的作图思路，然后了解作图函数所需要的数据结构，比如数据是什么类型、什么样的对应关系等，最后才是具体的实现细节。Sankey 图是一种变形的饼图或条形图，成功将数据进行流化，展示的数据面更宽阔。绘图所需数据包括三列：source、target、value。根据要研究的内容，Sankey 图可以分为三个层次：分子类型层、公司层和地域层，当然也可以根据需要扩展更多层次。第一步先筛出分子类型层：公司层对应的 sales，然后筛选出公司层：地域层对应的 sales，并与之前的 temp1 按照行合并为一个对话框，到这里数据就准备好了。

- 绘制 Sankey 图

```
1 sankeyPlot <- rCharts$new()
2 sankeyPlot$setLib("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取
/data/rCharts_d3_sankey-gh-pages/libraries/widgets/d3_sankey")
3 sankeyPlot$setTemplate(script = "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8
章数据抓取/data/rCharts_d3_sankey-gh-pages/libraries/widgets/d3_sankey/layouts/
chart.html")
4 sankeyPlot$set(
5   data = temp,
6   nodeWidth = 20,
7   nodePadding = 10,
8   layout = 10,
9   width = 1000,
10  height = 700
11 )
12 sankeyPlot
13 sankeyPlot$save('mychart1.html', cdn = TRUE)#保存为网页
```

首先开启一个新的 rchart 画板。然后设定调取库的路径和模块路径（刚开始下载的内容），通过 sankeyPlot\$set 设置所需的数据和参数，nodeWidth 设置节点条形宽度，nodePadding 设置节点条形的相对高度，越小越高，以及边框留白、整个图形的宽和高。最后执行 sankeyPlot 即绘制图形，通过 sankeyPlot\$save 可以将图形保存为网页格式。

从中可以分析出小分子药物市场地位依然举足轻重，罗氏的产品线多是创新型生物技术药物，而整个地域格局仍然是美国居主导地位，之所以小分子药物还那么强势，是因为大分子生产厂家很多没有包括在数据范围内，比如 2015 年美国最佳雇主排行榜 17 位的基因泰克（Genentech），即不在某家医药信息网站统计范围内。

其实投资股票，除了关注短期的资金流动以外，还要关注公司的基本面，比如医药公司主营销

售收入有多少，在同行中占据什么位置等，这都是作为投资者应该了解的。

## 2. 爬虫难道不需要模拟登录

有时候网站的信息并不能那么顺利地爬下来，比如有些网站需要注册登录，甚至会设置验证码等方法，从而限制爬虫抓取信息。当然，有句话说得好，“魔高一尺，道高一丈”，谁是魔谁是道，并不足以界定，只是在攻防双方存在一条魔道转化的底线。

正好某家医药信息网站也有注册登录通道，但是新注册客户和游客看到的信息比较相似，所以在之前的抓取程序中并没有模拟登录，为了给读者提供一个模拟登录的参考，这里就简单介绍一下使用 Cookies 模拟登录的过程。

Cookies 是网站服务器暂时存放在你电脑里的小资料，为 TXT 格式的文件，其作用就是帮助网站服务器识别你的计算机。在浏览网站时，网站服务器会先发送一个小材料（Cookies 文件）到你的计算机，Cookies 会记录下来你在网站上的搜索记录及选项等行为信息。当下次访问同一网站时，网站服务器会先寻找上次留下的 Cookies 资料，如果找到，就根据 Cookie 的内容做出判断，然后给你发送特定的网页内容。

Cookies 是长老级程序员 Lou Montulli 早期开发的用于计算机监控的技术“magic cookies”的简称，Ted Nelson（26 岁构思了 HTTP 基础架构，并以此创立公司）在 *Computer Lib* 书中讲了有关 Cookies 的起源的故事：

很久以前，一个大公司的账户系统是一个非常聪明的程序员编写和维护的，但他离开公司之后，奇怪的事情就发生了。该系统总是间歇性停运，并且控制台却显示一条消息：“给我一个 Cookie。”。只有输入了“Cookie”后，该系统才将重返常态。新管理员尝试了很多方法调试代码，但毫无建树，这种怪诞的行为总是无法除去，因为它是一段深埋代码，除了重写程序就不能完全被消除。所以最后决定最好还是保留代码，记录 Cookie 的问题，训练新操作员记得给机器一个 Cookie。

这个故事被程序员广为传赞，我们就不要揭穿本来就不太会讲笑话的程序员了，但他至少讲清楚了 Cookies 的作用。要使用 Cookies 模拟登录，首先要欺骗网站服务器将 Cookies 写入本机，这一过程叫作登录授权，即在真正的浏览器里访问网站；然后输入自己的用户密码，登录成功后将 Cookies 文件导出保存在 Cookies.txt 里，里面的内容大概以这样的格式摆放（为了防止我的账户泄密我删除了一部分内容），不要看具体内容，只要数据摆放格式正确就行（下面信息不需要执行），如下所示。

### • Cookies 格式

```
# HTTP Cookie File for domains related to druganalyst.com.
# This content may be downloaded or pasted into a cookies.txt file and used
by wget
# Example: wget -x --load-cookies cookies.txt http://consensus.druganalyst.
com/
#
1 consensus.druganalyst.com FALSE/FALSE0ASP.NET_SessionIdmt2sywkymjazto0y4rlnlg
2 .druganalyst.com TRUE / FALSE 144508155 _gat 1
3 consensus.druganalyst.com FALSE / FALSE 144536779.79049
```

```
4 .druganalyst.com TRUE / FALSE 150813581 _ga GA1.2.163822948.144962417
```

获得了 Cookies 文件之后，还需要设置一些参数模仿浏览器，就是使用 Rcurl 包中的函数模仿一个浏览器。浏览器（HTTP 客户程序）向网站服务器发送请求时需要指明请求类型，这些信息可以通过如下方式获得，然后按照相似的格式设置即可，在 Chrome 浏览器中按快捷键【Alt+Shift+I】查看头信息如下（不需要执行），选择“network”标签，刷新网页，找到该网页的 URL，点击后在右边选择 headers，就可以看到当前网页的 HTTP 头了。

- 浏览器抬头

```
1 Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
2 Accept-Encoding:gzip, deflate, sdch
3 Accept-Language:zh-CN,zh;q=0.8
4 Cache-Control:max-age=0
5 Connection:keep-alive
6 Cookie:ASP.NET_SessionId=t23m0engzliugddrovhgkuoc;
_ga=GA1.2.163822948.1444962417; _gat=1
7 Host:consensus.druganalyst.com
8 Referer:https://www.google.com.hk/
9 Upgrade-Insecure-Requests:1
10 User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/46.0.2490.71 Safari/537.36
```

具体参数指代的内容这里就不详细介绍了，只需要按照相应的名称填写在 myheader 向量中即可，如下所示。

- Cookies 模仿登录（1）

```
1 myheader<- c(
2 "User-Agent"="Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/46.0.2490.71 Safari/537.36",
3 "Accept"="text/html,application/xhtml+xml,application/xml;q=0.9,image/
webp,*/*;q=0.8",
4 "Accept-Language"="zh-CN,zh;q=0.8",
5 "Connection"="keep-alive",
6 "Accept-Encoding"="gzip, deflate, sdch"
7 )
```

很不幸地告诉大家，其实头信息不用设置也能成功登录，头信息模仿完成以后，就可以尝试登录了，debugGatherer 函数用于获得 Rcurl 提交给网站服务器的头信息等，可以通过 cat(debug\$value()[3]) 查看其中包含的信息，如下所示。

- Cookies 模仿登录（2）

```
1 debug <- debugGatherer()
2 curlHandle <- getCurlHandle(httpheader=myheader, debugfunction= debug$update,
verbose=TRUE, cookiefile="H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 8 章数据抓取
/data/cookies.txt")
3 temp<- getURL("http://consensus.druganalyst.com/Account/Login.aspx",
```

```
4 curl = curlHandle,.encoding="UTF-8")
5 html <- getURL("http://consensus.druganalyst.com/Guest/AbbVie/Humira",
curl= curlHandle)
6 grepl('5490728', html)
```

上述代码中，getCurlHandle 获得 URL 的句柄信息，以备加载 URL 时使用，这里的第 3 个参数 cookiefile 就是上一步登录授权后导出的 Cookies 文件（我的账号已经更改，最好自己注册一个），在 getURL 函数里添加 curl 参数，就能正式登录了。验证一下，比如加载一个药品销售状况的页面，网页下载完成后，可以在网页中查找到自己的用户名称，grepl 函数返回查找结果的 logic 值，如果查到，则为真，可以看到已经模拟登录成功了，反之则为假。

## 第 9 章

# 不可不说的社交网络关系

一个人可以没有社交吗？可以，但是没有社交的人，其基因不可能有效传给下一代，因为没配偶啊（是的，社交广泛的人也可能是单身）！人们将自己生活的点点滴滴公开在社交平台上，就少不了通过挖掘人们的社交数据来进行数据分析，特别是以市场研究为导向的数据分析挖掘，其实质就是研究人，而社交圈则深度反映一个人的生活状态和精神状态。

### 9.1 社交网络图

社交网络图谱（见图 9-1）不仅是一个人的关系网，而且代表信息的传播路径，找信息传播的高效路径很重要，你需要找到一条最短路径，找到最短路径就可以了吗？捷径一般比较拥堵，所以你还找到影响速度的短板，另外如今营销不仅要求覆盖的人数最多，还要求覆盖的人群尽量多样化，这些都和社交网络相关。更重要的是，它可能成为其他深度分析的基础，比如基于网络的推荐、分群、关联分析等。

#### 9.1.1 社交网络图告诉你和谁交朋友

你的朋友会不会把你变成胖子，或者朋友的身材和你个人的身材有关系吗？来自哈佛大学的研究者给出了肯定的答案。研究员 Christakis 在著名的《新英格兰医学杂志》撰文，认为肥胖具有社会传染性，也就是说肥胖是可以通过社交网络传染的，多么可怕，原来我的梨形身材是小伙伴传染的，友谊的小船估计又要翻了。

图 9-2 是一个 2200 人的关系网络，每个圆代表一个人，圆的大小代表肥胖指数的大小，不同颜色的圆圈表示不同的身材，线条代表社交关系。

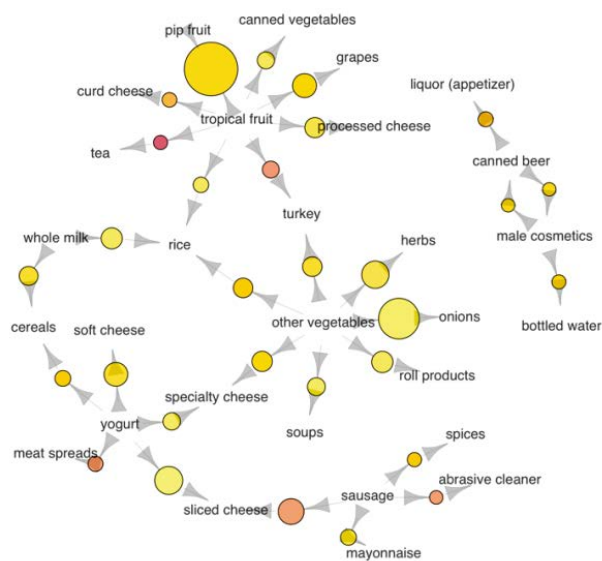


图 9-1

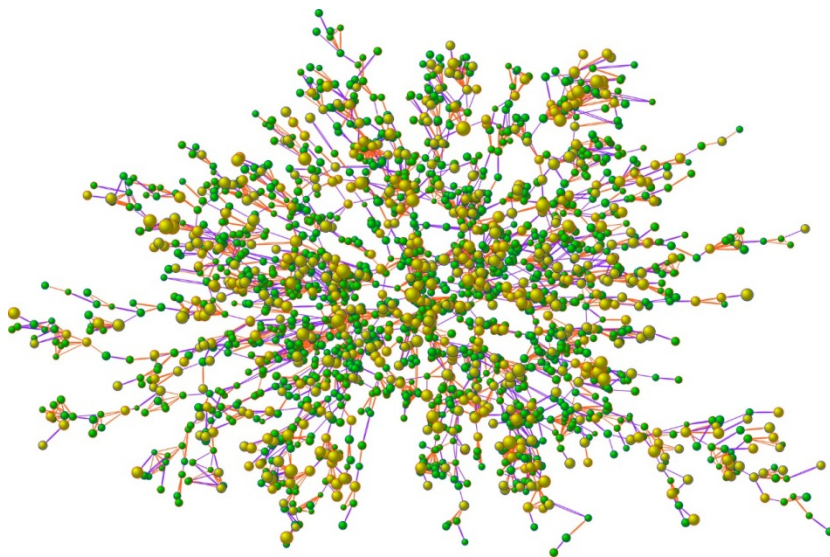


图 9-2

他们跟踪了 1 万多名受试者，花费了 32 年的时间研究，跟踪发现如果某人开始发福，和他关系近的人发胖的概率也会增加，而这里的“关系近”不是指血缘关系近，也不是指家庭关系近，而是社会关系近。如果你的好友发胖了，你发胖的概率将提升 57%，如果你是胖子的共同好友，你发胖的风险提高 171%，这些风险将要远配偶、邻居对你的影响大。

你是否觉得上面的结果不可思议，我也是这么认为，但是有人做过更有意思的关系网络研究，通过分析世界各国之间某货物的交易规模分析全球的各方贸易关系。

### 9.1.2 这几个基本概念你需要抓牢

其实我不喜欢聊概念性的东西，但是对于本章而言有些概念能够帮助大家更好地了解 igraph 对象及其所需要的输入格式，有时候有些名词翻译过来之后就不统一，了解概念也是为了统一理解一些英文名称。

#### 1. 社交网络的成分

社交网络是由节点 (node) 和边线 (edge) 组成的 (见图 9-2)，节点就是关系网络中的数据点，边线用来表示由数据点产生的关系的线条，这样通过边线将节点编制在一起就成了一张社交关系网络图。无论是节点还是边线都是我们后面研究的基本对象，比如想知道网络中的中介人是谁？

#### 2. 社交网络的类型

由节点和边线这两个基本元素组成的网络类型五花八门，但就关系的方向性而言，网络关系类型至少可以分为两种：指向型和非指向型。网络中的方向性往往代表着信息、能量等的流向或传播路径。

指向型关系网络表示网络中的关系是有方向性的，这种方向性在单方向暂时不可逆，比如说在微博上，你关注了“女神”，“女神”却没关注你，从主动性方面看，你们的关系应该从你指向“女神”，但是从信息流动的方向看是“女神”指向你，因为你可以接收到“女神”发布的任何信息，而“女神”却收不到你的日常，但是无论是从主动性还是信息流动上看这种关系都是指向型关系，有这种指向型关系组成的网络就是指向型关系网络（如下例）。

##### • 指向型关系网络示例

```
1 library(igraph)
2 asymmetrical <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的社交网络关系/data/asymmetrical.csv", header=TRUE, row.names = 1)
3 asymmetrical <- as.matrix(asymmetrical)
#           孙红雷 刘诗诗 霍建华 刘亦菲 张嘉译
# 孙红雷      0      2      3      0      3
# 刘诗诗      1      0      5      2      2
# 霍建华      2      6      0      1      1
# 刘亦菲      1      2      0      0      0
# 张嘉译      4      1      1      1      0
4 asymmetricalnet <- graph.adjacency(asymmetrical, mode = "directed", weighted
= TRUE)
5 asymmetricalnet
# IGRAPH DNW- 5 17 --
# + attr: name (v/c), weight (e/n)
```



上述代码中,第2行代码读取数据, `row.names` 将数据第1行设置为行名称;第3行代码将数据转化为相邻矩阵。相邻矩阵用于表示变量(对象)之间距离关系的矩阵,它的行和列代表着不同的节点, `graph` 将节点之间的相邻矩阵转化为 `igraph` 对象之后作图。指向型关系网络一般会存在关系强度的不对称性,也就是说节点的相邻矩阵为非对称,例如此例中孙红雷转发刘诗诗的微博次数和刘诗诗转发孙红雷的微博次数是不相同的,也就是说这种转发关系是不对称的。如果是非指向型关系网络,相邻矩阵为对称的,比如刘诗诗和孙红雷的合作次数与孙红雷和刘诗诗的合作次数肯定是相等的(参见本书数据集里的 `symmetrical`),这种关系具有对称性。第4行代码使用 `graph.adjacency` 函数将相邻矩阵转化为 `igraph` 对象,这个函数接受的必须是行和列为节点名称,每个格子里的数字表示关系强度(或有无)的相邻矩阵, `mode` 参数指定指向型(`directed`)或非指向型网络(`undirected`), `weighted` 参数设置是否把相邻矩阵的数值设置为权重。

建模或绘图时可以什么都不会,但是必须首先知道函数的输入输出是什么类型的对象,简单说即明白数据是怎么摆放的。

查看 `asymmetricalnet` 函数,输出结果的第1行代码表示创建了一个 `IGRAPH` 对象,第1个字符 `D` (`Directed/Undirected`) 表示指向型网路,第2个字符 `N` (`Named`) 表示节点已经有名字属性了,如果没有则用短横线“-”表示,第3个字符表示关系网络是不是加权了,如果加权就会用“`W`”( `weighted` )表示,非加权就用短横线表示,第4个字符表示是否是二分式的网络(`Bipartite`),所谓二分式就是节点之间有层次关系的,也就是说节点只能和它的下级节点建立关系,平级节点之间不存在关系,数字表示共有5个节点,17条边线(关系);第2行代码表示这个关系网络的节点拥有哪些属性,这里的节点拥有名称和权重属性,后面我们的节点可能还有性别、颜色等属性(属性随便加),名称属性的 `v` 表示节点(`vertex`), `c` 表示名称属性的数据类型为 `character`,当你要提取名称属性时,需要以 `V(asymmetricalnet)$name` 形式提取,权重属性 `e` 表示边线(`edge`), `n` 表示权重的数据类型为整数,当你要提取权重属性时,需要以 `E(asymmetricalnet)$weight` 形式提取。

#### • 绘制指向型关系网络

```
1 require("extrafont")
2 loadfonts(device="win")
3 plot(asymmetricalnet, edge.width = E(asymmetricalnet)$weight, edge.arrow.size = 0.1, vertex.size = 9, vertex.label.dist = 0.55, vertex.label.cex = 0.8, edge.curved = TRUE)
4 title(main = "微博转发关系指向型网络", cex.main = 2, col.main = rgb(64, 64, 64, max = 255), family = "STXinwei")
```

上述代码中,使用基础函数 `plot` 绘制网络关系图,当然也可以有很多个性化参数可以设置(第3章已经讲过一些), `edge.width` 即边线粗细,将其映射为边线的权重;后面几个参数分别用于指定箭头大小(`edge.arrow.size`)、节点大小(`vertex.size`)、节点标签的移动距离(`vertex.label.dist`)、节点标签的字体大小(`vertex.label.cex`), `edge.curved` 防止关系边线重合,以曲线形式显示边线;最后一行代码使用 `title` 函数给关系网络图添加标题。

效果如图9-3所示。

微博转发关系指向型网络

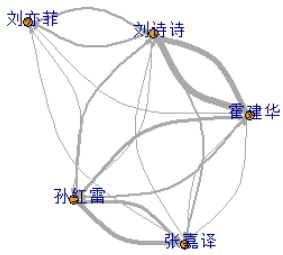


图 9-3

非指向型关系网络就比较容易理解了，关系不具有方向性，比如购物篮里的商品，虽然都是同一次购进的，但并不代表这种关系具有方向性，不进行分析或测试还真的很难分清是从啤酒指向尿布还是从尿布指向啤酒，甚至到底有没有这种指向关系存在都是值得怀疑的，因此这种关系可以归为非指向型关系。

我们这个行业也逐渐混进了一批讲故事的人，原以为这类人只有在传销团队才能生活，但是他们确实使用自己的嘴骗了一批又一批的数据分析师的学费。

• 非指向型关系网络示例

```
1 symmetrical <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的社交网络关系/data/symmetrical.csv", header=TRUE, row.names=1)
2 symmetrical <- as.matrix(symmetrical)
#      孙红雷 刘诗诗 霍建华 刘亦菲 张嘉译
# 孙红雷      0      2      3      0      3
# 刘诗诗      2      0      5      2      2
# 霍建华      3      5      0      1      1
# 刘亦菲      0      2      1      0      0
# 张嘉译      3      2      1      0      0
3 symmetricalnet <- graph.adjacency(symmetrical, mode = "undirected",
weighted = TRUE)
4 symmetricalnet
# IGRAPH UNW- 5 8 --
# + attr: name (v/c), weight (e/n)
```

上述代码中，第 1 行代码读取数据；第 2 行代码将数据框转化为对称型相邻矩阵，我们可以看到数据是沿对角线对称的；第 3 行代码使用 graph.adjacency 函数将相邻矩阵转化为 igraph 对象；第 4 行代码查看对象的基本信息。

上面我们已经熟悉了将相邻矩阵转化为 igraph 对象的过程，其实 igraph 接受的另一种数据类型为关系列表 (edge list)，关系列表至少包含两列，分别是用于组建关系网络的节点，如果有关系权重可以作为第三列，其实关系列表和相邻矩阵可以相互转换，只是分析时数据输入的方式不同而已，后面我们将重点讲解。

- 绘制指向型关系网络图

```
1 plot(symmetricalnet, edge.width = E(symmetricalnet)$weight, vertex.size =
9, vertex.label.dist = 0.55, vertex.label.cex = 0.8)
2 title(main = "影视合作关系非指向型网络", cex.main = 2, col.main = rgb(64, 64,
64, max = 255), family = "STXinwei")
```

效果如图 9-4 所示。

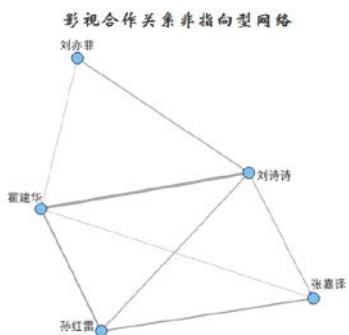


图 9-4

根据节点类型又可以划分为从属性关系网络、非从属性关系网络和混合网络，从属性关系网络表示节点之间存在严格的从属关系，大部分社会类关系网络属于这种类型，而微博的关系网络就是非从属性关系网络，大 V 和粉丝之间并没有严格的阶级关系，只是一种简单的相互关系。更多时候我们的社会关系是这两种网络共同构成的混合网络，在公司中我们有部门之间、上下级之间的从属关系，在社交媒体上我们和范冰冰之间虽然存在关注和被关注的关系，但是并没有严格的上下级关系。

### 9.1.3 还有比本章任务更有趣的数据挖掘吗

这一章的重要性不用多言，其实我个人认为行业里的任何一个点都是职业人不能忽视的，但作为要成为一个行业的专家的人，只强调行业的任何一点的重要性是没有意义的，因为在他们眼里看待一点就像看待佛珠一样，缺少任意一个环节都不是完美的。

成不了龙也要成虫儿。

——老北京俗语

除了其重要性以外，这章要回答一些有趣的问题，首先我们需要了解一下社交网络的一些基本知识，在此基础上我们要完成两个案例分析：

(1) 全球某货物贸易分析，可能要回答以下问题：

① 全球某货物贸易的网络长啥样？

② 全球某货物贸易网络可以分几个小圈子？

③ 哪些国家是某货物交易的中心？这里我们学习 PageRank 算法在网络分析中的应用，找出全球各国之间的关系，以及这些关系的特征。

(2) 中国演艺圈分析，我们试图回答下面这些问题：

① 中国的演艺圈到底存不存在？

② 众多演员中谁的合作圈最强悍？

③ 怎么通过算法发现演艺圈的好朋友？

④ 能不能给范冰冰推荐一个合作伙伴？

是不是很好玩儿，不管你信不信，反正我觉得这章最好玩儿，而且还可以锻炼一个人的逻辑分析能力。

## 9.2 你还要装备几个评价指标

这一节需要了解一些衡量网络大小、完备性、节点实力的指标，这些指标是比较两个截然不同的网络时必选的方式，同样也为衡量信息传播路径提供侧写，比如要筛选大 V 做营销传播时，除了大 V 粉丝和营销事项的契合度以外，还要进一步评价他的信息传播路径，那么比较网络圈子就成了不可避免的事情。

### 9.2.1 社交网络大小

比较两个圈子，首先从整体上着手，节点数和边线数是直接反映整体的指标，节点数表示这个圈子总共编制了多少终端（人或物），边线数表示圈子创造了关系。节点衡量信息最终会被多少终端接收到，边线衡量信息传播的路线有多少。上面我们已经查看了一个网络的节点数和边线数，下面的代码可以查看具体的节点和边线。

- 节点和边线

```
1 V(symmetricalnet)
2 V(symmetricalnet)$name
3 E(symmetricalnet)
4 get.data.frame(symmetricalnet)
```

上述代码中，通过第 1 行代码可以查看每一个节点的名称；第 2 行代码可以将 igraph 对象的节点转化为 vector；第 3 行代码查看每一条关系记录；第 4 行代码将关系转化为数据框，默认状态下数据框包括关系的权重。通过节点数和关系数能够帮助我们对一个关系网络有一个大概印象。

## 9.2.2 社交网络关系的完备性

完备性考察一个网络的复杂性，有时候一个网络关系的复杂程度并不能简单地通过节点数评价，比如一串佛珠的关系网络并不复杂，因为每个佛珠只和相邻的两个佛珠形成了一步关系，所以需要更加深入恰当的指标评价网络的复杂性，比较常见的指标包括社交网络密度和社交网络连通性两类指标。

### 1. 社交网络密度

你要给领导这样解释社交网络密度：一个网络的密度就是对这个网络的完备性的一种测度，它在一定程度上象征着这个网络中关系的数量与复杂程度。换成数据分析师之间的语言：网络密度等于一个网络的关系数除以这个网络理论上的关系数，密度最大值为 1，即每个个体都与其他所有个体产生关系。

- 网络密度

```
graph.density(symmetricalnet)
```

有些研究指出，在现实生活中，网络中能够发现的最大密度是 0.5，而人人网的网络密度值在 0.1 左右。网络的密度越高，信息传达的路径越多，单个路径的信息压力越小。

### 2. 社交网络连通性

有时候不能强求网络的密度达到很高的程度。牛津大学的心理学家罗宾·邓巴研究发现，人类所能维持的社交圈一般为 150~200 人左右，包括亲朋好友，即便你疯狂地在社交媒体上添加好友，真正能够与你沟通交流的人群一般不会超过这个数字，这一数字被命名为邓巴数字，更加令人沮丧的是我们的“死党”一般为 5 人，所谓志同道合的密友也不会超过 15 个，多么令人尴尬的现实，因为灵长类大脑皮层的限制，你只有那么多时间和精力去维持这个小圈子，文艺一点的说法就是这正是很多朋友慢慢地走出你视野的原因。

如果两个网络的密度都不高，是不是就没有办法进一步评价了呢？考量社交网络的可达性可以帮助我们缓解这个问题，网络中所有个体通过多少个步骤相互联络在一起，也就是网络的连通性，测量连通性的指标为直径（diameter）。所谓直径表示网络中任意可连通的两点之间的最大距离。直径越短，表示可以通过很少的步骤访问完整个网络。

- 网络直径

```
1 diameter(symmetricalnet)
2 get_diameter(symmetricalnet)
```

上述代码中，第 1 行代码返回网络的直径；第 2 行代码返回最大距离等于直径的一条关系链，可能存在多条，这里只展示第一条。其实可以通过 NetIndices 包的 GenInd 函数，一次性输出有关某个网络的 10 个常见的指标，如下所示。

- 输出所有基本表

```
1 library(NetIndices)
2 test.graph.adj <- get.adjacency(symmetricalnet, sparse=F)
```

```
3 test.graph.properties <- GenInd(test.graph.adj)
4 test.graph.properties$LD # 平均每个节点的关系数（边线）
5 test.graph.properties$C # 网络连接度
```

上述代码中，第 2 行代码通过 `get.adjacency` 函数提取社交网络的相邻矩阵（网络比较大时，效率比较低），然后通过相邻矩阵计算常见的 10 个指标，重点说一下，`test.graph.properties$C` 即为上面所说的社交网络密度。

### 9.2.3 节点实力评价

在现实工作中，特别是在企业应用中，可能关注网络中的个体更多一些，而不是像研究机构从网络整体到个体都加以关注研究，比如企业在评价代言人、筛选草根大 V 进行营销活动时，都是更加注重这些网络中的个体。

#### 1. 度数（点度中心度）

所谓点度中心度（Degree Centrality）就是与该点直接相连的点的个数。度数越大表示节点在网络信息传播的过程中权重越大（非指向型关系）。

- 相邻节点

```
DC <- length(neighbors(asymmetricalnet, "孙红雷"))
```

但需要提醒的是，如果两个节点之间产生了多次关系，则 `neighbors` 会重复计数，比如孙红雷和霍建华曾经合作过两次，那么孙红雷的相邻节点里就会出现两次霍建华，所以如果你的网络关系是加权的或指向型网络，对相邻节点计数时不要忘记去重。

#### 2. 关系总数

一个节点的重要性除了看它的邻居总数以外，还要看它产生的关系总数。

- 相邻节点

```
incident(symmetricalnet, "孙红雷", mode=c("total"))
```

上面获取了孙红雷产生的合作关系的总数，在指向型关系网络中关系又分为进关系（in）和出关系（out）等，对不同类型的关系计数可以使用 `mode` 设置，`incident` 函数生成的是一个包含关系的向量，需要使用 `length` 函数对其结果计数。

进关系和出关系在信息传播的过程中意义是不同的，进关系代表着节点信息获取的能力，出关系代表节点信息搜集的能力，当然这不一定是绝对的，看你怎么定义进关系和出关系，但是无论怎么定义，它们在信息传播的过程中所起的所用绝对不同。

#### 3. 中介度（Betweenness Centrality）

简单地讲，中介度是指所有的节点对之间通过该节点的最短路径条数，`betweenness` 函数定义的中介度为：一个节点的中介度表示通过该节点的任意两个节点之间的最短路径条数除以这两个节点之间最短路径总数之商的连加和（不懂可以看帮助文档）。中介度很好地描述了一个网络中节点可能

需要承载的流量。某节点的中介度越大，流经它的信息流越多，正如六步分割原理（Six Degrees of Separation）所说的一样，并不是说我们每一个人仅仅经过六步就可以和世界上任何人联系在一起，而是说世界上有些人和其他任意一个人联系在一起不会超过六步，而我们可以通过和这些人认识，就可以通过他们与其他人联系在一起了，而这些人就是我们所说的中介度非常高的节点。是不是很神奇，我们的社交圈那么小，而要想和其他人联系起来又那么简单，只需要通过几个社交达人就可以了。

```
betweenness(symmetricalnet)
```

betweenness 函数计算网络中所有点的中介度，我们可以通过选择中介度较高的几个人作为信息传播的重要节点，因为他们就是《引爆点》中所说的一条信息爆炸传播中不可或缺的三种人之一——联系员，其他两种为内行（达人）和推销员，也就说由知识丰富的内行制造一条信息然后由推销员将信息转化为人易于接受的形式传给关系很广的联络员并由其散播，一条信息必将由这样的路径引爆。

除了上述的简单评价指标以外，对于复杂网络关系，我们可能需要综合性的算法评价节点的重要性，比如排名算法中著名的 PageRank、HITS 等。

## 9.3 全球某货物贸易中的亲密关系

本小节是受一篇国外的博文启发，试想从全球某货物交易圈看看全球各国之间的关系。分析这些交易情况可以帮助我们找到好伙伴，首先从全球某货物数据库（SIPRI）下载 2000—2015 年的交易记录，SIPRI 是一个很有意思的数据库，很多交易记录都可以非常方便地下载。

### 9.3.1 全球某货物贸易数据整合清洗

按照进口国家下载某货物交易的数据，大概包括 185 个 CSV 文件，文件记录了该国历年来从哪些国家进口了某货物，例如文件名称 TIV-Import-AFG-2000-2015，其中 AFG 是阿富汗的缩写，表示 2000—2015 年阿富汗的某货物进口记录，我们需要将文件一个一个地批量读取，然后经过适当转换将它们整合为一个某货物交易记录表。

#### 1. 全球某货物贸易数据整合

一位“牛人”说数据挖掘工作 80% 的时间用于数据清洗，另外 20% 的时间用于抱怨数据清洗，这话说得非常中肯，不信可以统计本书的代码，大部分工作是在做数据清洗、整合和指标构建，数据建模的工作几乎没有几行代码，所以我个人建议，学习挖掘工具重在数据清洗，学习挖掘算法重在统计理论和数学，此乃内外双修的要点。同样我们仍然需要先对交易记录做一番清洗整合工作。

- 交易记录清洗

```
1 library(reshape2)
2 weaponpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的社交网络关
```

```

系/data/weapon"
  3 completepath <- list.files(weaponpath, pattern = "*.csv$", full.names =
TRUE)
  4 dataclean <- function(x) {
  5   data <- read.csv(x)
  6   data <- data[, -1]
  7   data <- data[-c(1:9), ]
  8   temp <- unlist(data[1,])
  9   temp[1] <- "Export"
 10   temp[length(temp)] <- "total"
 11   names(data) <- temp
 12   data <- data[-1,]
 13   data <- melt(data, id = "Export")
 14   data <- data[!is.na(data$value),]
 15   data <- data[!data$variable == "total",]
 16   data <- data[!data$Export == "Total",]
 17   temp <- unlist(strsplit(x, "/"))
 18   temp <- temp[length(temp)]
 19   temp <- unlist(strsplit(temp, "-"))
 20   Abbr <- rep(temp[3], length(data[,1]))
 21   data <- cbind(Abbr, data)
 22 }
 23 weapon <- lapply(completepath, dataclean)
 24 weapon <- do.call("rbind", weapon)
 25 names(weapon) <- c("Abbr", "Export", "year", "dollar_m")

```

上述代码中，第 2 行代码设定数据文件路径；第 3 行代码的 `list.files` 函数获取该路径下所有文件的文件名，如果 `full.names` 参数为真，则返回完整的路径，反之返回文件名称，`pattern` 设置符合正则表达式的规则，仅提取符合要求的文件，防止读入系统文件，这里仅匹配以.csv 结尾的文件；第 4 行代码起是一个自编函数 `dataclean`，用于清洗整理数据，函数仅有一个参数，就是文件的完整路径，第 5 行代码读取路径 `x` 下的文件，因为文件的第一列都是一些说明文字（可以用 Excel 打开查看），所以第 6 行代码将第一列删除；第 7 行代码用于删除文件的前 9 行，因为前 9 行都是空行，这些操作完成之后我们就需要给原来的数据列重新命名，删除后的第 1 行数据可以作为列名，即把年份作为列名；第 8 行代码提取了第 1 行数据并 `unlist`，但是数据的第一列和最后一列数据和年份无关，第一列是出口某货物的国家名称，最后一列是历年汇总；第 9 行和第 10 行代码将 `temp` 的第一个和最后一个元素分别更改为 `Export` 和 `total` 列；第 11 行代码按照 `temp` 重命名原来的数据框 `data`；第 12 行代码删除原数据的第 1 行（年份）；第 13 行代码使用 `melt` 函数将数据从 `wide` 型转化为 `long` 型；第 14 行代码使用 `is.na` 函数删除没有交易数据的年份；第 15 行和第 16 行代码删除包含年份汇总的数据行；第 17 行代码使用 `strsplit` 函数将 `x` 路径按照 “/” 分裂，然后将 `unlist` 赋值给 `temp`；第 18 行代码取 `temp` 最后一个元素，也就是文件名，因为文件名 `TIV-Import-AFG-2000-2015` 包含着武器进口国家名称缩写；第 19 行代码按照 “-” 再将 `temp` 分裂，将 `unlist` 解散后赋值给 `temp`；第 20 行代码



取 temp 第 3 个元素即进口国缩写，将缩写复制和 data 等长；第 21 行代码将其和 data 按列捆绑在一起。

倒数第 3 行代码使用 lapply 函数将所有的文件路径传递给自编函数 dataclean，执行获得一个 list；倒数第 2 行代码将结果按行粘贴为数据框，赋值给 weapon，然后对 weapon 各列重新命名，重命名的函数我们已经总结过一次了，另外需要提醒的是货币计量单位为百万美元。

## 2. 全球某货物贸易数据清洗

花时间做好每一个看似没有必要的细节就是作为小职员的分内之事。

我们不能直接在图上展示各国的英文名，虽然大部分人能看懂英文，但作为职场中的我们，必须从每一个细节体现自己的专业水平和用心程度，因此这里需要将进出口国家的英文名称替换为中文，我们从“ISO\_3166\_1”全球国家名称统一缩写文件中整理出一个对应的英汉对照版本，然后将它们进行匹配即可，如下所示。

- 英译汉

```
1 library(plyr)
2 wcountry <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的
社交网络关系/data/wcountry.csv", header=TRUE)
3 temp <- wcountry[, c("Abbr", "chinese")]
4 weapon <- join(weapon, temp)
5 weapon <- rename(weapon, replace = c("chinese" = "import"))
6 temp <- wcountry[, c("country", "chinese")]
7 weapon <- rename(weapon, replace = c("Export" = "country"))
8 weapon <- join(weapon, temp)
9 weapon <- rename(weapon, replace = c("chinese" = "export"))
10 weapon <- weapon[!is.na(weapon$import),]
11 weapon <- weapon[!is.na(weapon$export),]
12 weapon <- weapon[, c("export", "import", "year", "dollar_m")]
```

上述代码中，第 2 行代码读取英汉对照文件；第 3 行代码提取缩写列和中文列；第 4 行代码使用 join 函数按照进口国家的缩写给进口国家匹配上中文名称；第 5 行代码使用 rename 函数将刚匹配上的 chinese 列改名为 import 列；第 6 行代码提取 country 列和 chinese 列赋值给 temp；第 7 行代码将 weapon 数据框中的出口国家列名称 Export 改为 country；第 8 行代码按照 country 列匹配 weapon 和 temp，这样就给出口国家列匹配上了相应的中文；第 9 行代码将刚匹配上的 chinese 列改名为 export，即出口国家列；第 10 行代码移除进口国家为空的数据；第 11 行代码移除出口国家为空的数据；第 12 行代码筛选出下面将要用到的数据列，包括某货物出口国、某货物进口国、年份、金额。

- 透视表

```
1 weapon$dollar_m <- as.numeric(weapon$dollar_m)
2 weapon <- aggregate(dollar_m ~ export + import, weapon, sum)
3 exvsim <- weapon
```

上述代码中，第 1 行代码将 dollar\_m 列由字符型变量转化为数值型变量，注意，在 R 里不要将

因子直接转化为数值型变量，需要先将因子型变量转化为字符型变量，然后再转化为数值型变量，否则得不到你想要的结果；第 2 行代码将交易额按照出口国和进口国分组求和，做透视表就统计出了 2000—2015 年这 16 年里各国某货物进出口的交易总额；第 3 行代码将 `weapon` 赋值给一个新的数据框 `exvsim`，留存用于分析进出口的差异。

完成以上清洗还需要对数据做最后一拨处理，我们要研究两个国家之间的总交易额，但是根据以上处理并不能完整汇总得出两国的交易总额，比如 a 国出口到 b 国总计 100 万美元的某货物，但同时从 b 国进口了 200 万美元的某货物，记录表里是两条数据，而我们需要这两条数据数据的总和（如下例）。

```
1 weapon[c(39, 1144),]
#      export import dollar_m
# 39   西班牙  阿根廷      18
# 1144 阿根廷  西班牙       6
```

那就需要找出这类记录，并将它们加和为两国交易的总和，而不是细分为进出口，当然如果你要进一步研究，也可以作为指向型网络研究。我们这里使用集合的方法来处理这种 a、b 和 b、a 的问题，即如果两个集合元素相同，将其中一个找出来逆转一下就可以了，之所以这里用到这种特殊方法，是想让大家了解到，当需要处理一个两个字符串的相似性（非语义）时，可以从集合的角度思考，使用 jaccard 系数解决问题。

- 指向记录转为非指向记录

```
1 library(sets)
2 temp1 <- weapon[, c("export", "import")]
3 dupli <- data.frame()
4 for (i in 1:length(temp1[,1])) {
5   m <- as.set(as.character(unlist(temp1[i,])))
6   n <- i+1
7   for (j in n:length(temp1[,1])) {
8     s <- as.set(as.character(unlist(temp1[j,])))
9     sim <- set_similarity(m, s)
10    if (sim == 1) dupli <- rbind(dupli, j)
11  }
12 }
13 temp2 <- weapon[dupli$X1144L, c("import", "export", "dollar_m")]
14 names(temp2) <- c("export", "import", "dollar_m")
15 temp1 <- weapon[-dupli$X1144L, ]
16 weapon <- rbind(temp1, temp2)
17 weapon <- aggregate(dollar_m ~ export + import, weapon, sum)
```

上述代码中，第 2 行代码提取 `export`、`import` 列备用；第 3 行代码创建一个空数据框，用于接收找到的相似行编号；第 4 行代码开始一个外部循环，`temp1` 有多长就循环多少次；第 5 行代码将 `temp` 第 `i` 行提取出来，`unlist` 之后变为向量，然后将其中的两个国家名称转化为字符（`as.character`），再将字符转化为一个集合 `m`（`as.set`）；第 6 行代码将 `i` 加 1 赋值给 `n`；第 7 行代码从 `temp1` 的第 `n` 行开始

第二层循环,也就是说从  $i$  加 1 行开始寻找和  $i$  行相似的行;同样第 8 行代码将第  $j$  行转化为集合  $s$ ;第 9 行代码使用 `set_similarity` 函数计算  $m$  和  $s$  的相似性,默认状态下使用 jaccard 系数衡量相似性;第 10 行代码判断两个集合的相似性是否为 1,如果为 1 表示两个集合的元素相同,将其行编号  $j$  和数据框按行捆绑在一起即可;第 13 行代码按照 `dupli` 的行编号提取行,先提取“import”列,然后提取“export”列,最后提取金额赋值给 `temp2`,这样 `ba` 就变成了 `ab`,不仅元素相同,顺序也相同了;第 14 行代码使用 `names` 函数将 `temp2` 重新命名,这样“import”列被命名为“export”,而“export”被命名为“import”;第 15 行代码删除重复行并赋值给 `temp1`;第 16 行代码将 `temp1` 和 `temp2` 重新 `rbind` 在一起,数据就调整好了;最后一行代码再做一次透视加和即可。

上述方法应该是最简单的方案,说它简单是因为它的循环思维方式最容易想到,但是速度太慢了,遇到巨量数据可能要花费很长一段时间,后面我们将要设计更加高效的方法。

### 9.3.2 分组和社交网络中心

分析社交网络的关系,最终需要回答一个圈子里有哪些群体和这些群体的中心点的问题,应用到我们的分析课题里就是要找出某货物交易有哪些小圈子,这些小圈子里有哪些人物。首先要对某货物交易圈分组,然后再找出某货物交易中的一些中心点,也就基本上回答了上述问题。

#### 1. 网络成员分组：分组算法比较

分组就是站队,站错队轻则加重机会成本,重则随队长一起被灭亡。社交网络的分组算法有很多,但分组算法除了研究群体之间的相似性,还有其在数据可视化领域的利用。如果一张社交网络图有上千万个数据点,自然需要分组算法层层展示,否则绘制一次就会消耗大量的内存和时间。

##### • 分组算法比较

算法	Directed edges	Weighted edges
Edge-Betweenness	是	是
Leading Eigenvector	否	否
Fast-Greedy	否	是
Multi-Level	否	是
Walktrap	否	是
Label Propagation	否	是
InfoMAP	否	是

进行分组的算法有很多种,其速度和能够处理的网络类型各不相同,有些仅能处理非指向、非加权的网络,有些能做到加权和指向型双吃,但是无论选择哪一种算法,请记住它们的共同特征,都是基于节点和关系来组织计算分组的,所以我不建议使用一门学问解决所有的问题。比如分组,如果将自己限制在网络分组算法的胡同里,可能永远也得不到好的分组,因为组员之间的相似性信息仅仅来自节点和他们之间的关系。但是试想一下,我们学了多少有关分组的东西,统计学的一个大问题就是分类。因此,如果将问题换为分类或聚类,那么能用到的信息和算法会更多,除了节点之间的关系,我们还可以找到节点一些属性的相似性,比如这里要考虑两个国家是否分在一组,可

能仅仅根据他们的关系判断，但是若换成聚类的思维，可能还会使用国家之间的经济、文化、教育等变量之间的相似性完成分组，所以具体使用哪种方法一定要非常细心地分析业务问题，而不是将自己限制在某一方法内。这里采用 Multi-Level 分组算法对某货物交易圈分组，Multi-Level 分组算法是一种基于模式识别和层次分类的方法，测试之后效果最好，注意这个效果好不好只是个人感觉。

- 网络分组

```
1 weapon <- weapon[weapon$dollar_m > 200,]
2 weaponnet <- graph.edgelist(as.matrix(weapon[,1:2]), directed = F)
3 E(weaponnet)$weight <- weapon[,3]
4 community <- multilevel.community(weaponnet, weights = E(weaponnet)
$weight)
5 subgroup <- split(V(weaponnet), community$membership)
6 subgroup[1]
7 V(weaponnet)$sg <- community$membership
```

上述代码中，第 1 行代码提取双方交易额大于两亿美元的国家；第 2 行代码提取 weapon 的前两列，as.matrix 将其转化为矩阵，也就是一个关系列表，你可以通过 as.matrix(weapon[,1:2])查看关系列表长什么样，前面已经介绍了 igraph 可以接受相邻矩阵，也可以接受关系列表（edge list），然后通过相应的函数将其转化为 igraph 对象，这里使用的是 graph.edgelist 函数，我们将网络类型指定为非指向型；第 3 行代码将 weapon 的第三列作为关系权重赋值给网络对象 weaponnet，这样 igraph 网络对象就构建完成了；第 4 行代码使用 multilevel.community 函数对网络节点分组，通过 weights 指定权重；第 5 行代码使用 split 函数将第 1 个参数按照第 2 个参数分组，其结果就是一个 list，但是两个参数必须是等长的，V(weaponnet)提取网络节点名称，community\$membership 提取节点的分组编号；第 6 行代码查看第一个小组成员；第 7 行代码将分组编号赋值给节点属性 sg(subgroup)，记住，你可以通过这种方式不断地给节点或边线赋各种属性，如颜色、线等。

通过分组不仅可以了解到这个某货物交易圈有哪些小团体，而且也发现各国的贸易关系。

## 2. 使用 PageRank 计算政治中心

前面第 7 章已经讲解了 PageRank 算法，简单点解释，一个节点的重要性的指向它的节点数及指向它的节点的重要性相关，我们就使用 PageRank 算法算一算某货物交易圈里各个节点的重要性，找一找各个群体的“带头大哥”是谁。

- PageRank 计算政治中心

```
1 pagesize <- page.rank(weaponnet, algo = "prpack", vids = V(weaponnet),
directed = F)
2 pagesize <- pagesize$vector
3 V(weaponnet)$size <- pagesize
```

上述代码中，第 1 行代码使用 page.rank 函数计算节点的重要性，第 1 个参数为 igraph 网络对象，第 2 个参数 algo 用于设定调用的算法库，这里默认 prpack 库，它处理大型网络速度快而且比较稳定，vids 指定你感兴趣的节点名称，directed 参数用于指定是否是指向型网络；第 2 行代码提取计算结果，结果中的 vector 包含每个节点的重要性；第 3 行代码将结果赋值给节点的 size 属性，可视化时用于

映射节点的大小。

你可以使用这个方法构建一个自动推荐 R 包的脚本 ( <https://gist.github.com/andrie/73f23d15fed2f82853c1> )。

### 3. igraph 怎么调节每一个节点的位置

数据可视化其实就是数据到图形属性的映射，要展示网络结构首先要确定网络节点的位置，即布局 ( layout )，这就需要算法 ( 即映射关系 ) 支撑，也可以随机地给每一个节点在图中找到一个位置坐标，可以按需求将节点摆出各种形状，这些从简单到复杂的位置坐标需求都需要相应的算法支持。

这里也需要根据某种算法确定节点位置，并保证统一分组的节点尽量聚在一起，igraph 中支持了很多布局算法，最常用的为 layout.fruchterman.reingold，但是它对我们的问题比较无力，很多群组挤在一起，所以我们使用了 gephi ( 复杂网络分析软件 ) 的一种布局算法：ForceAtlas。

- 安装

```
1 library("devtools")
2 install_github("analyxcompany/ForceAtlas2")
3 library("ForceAtlas2")
```

ForceAtlas 还没有正式在 R 官方源上发布，只能从 Github 上下载安装，在线安装 Github 的包需要安装 Rtools，并同时安装 devtools 包才能进行，当然你也可以将包下载下来进行本地安装，如下所示。

- igraph 怎么确定每一个节点的位置

```
1 set.seed(1639)
2 layout <- layout.forceatlas2(weaponnet, iterations = 200000, plotstep = 500)
#layout <- layout.fruchterman.reingold(weaponnet)
3 head(layout)
4 plot(weaponnet, layout = layout, vertex.size = 4)
5 layout[V(weaponnet)$sg == 1, 1] <- layout[V(weaponnet)$sg == 1, 1] + 100
6 layout[V(weaponnet)$sg == 1, 2] <- layout[V(weaponnet)$sg == 1, 2] - 100
7 plot(weaponnet, layout = layout, vertex.size = 4)
8 layout[79, 1] <- layout[79, 1] - 5 ###调节印度位置
9 layout[79, 2] <- layout[79, 2] + 5 ###调节印度位置
# layout[5, 2] <- layout[5, 2] - 3 ###调节中国位置
```

因为 layout.forceatlas2 函数有一个随机开始的迭代过程，所以为了保证随机数的一致性，上述代码中的第 1 行代码设置了随机种子；第 2 行代码使用 layout.forceatlas2 函数计算布局，iterations 设置迭代次数，plotstep 设置每隔多少次迭代绘制一次迭代结果；第 3 行代码查看布局结果，结果是一个矩阵，第一列代表节点的 X 轴坐标，第二列代表节点的 Y 轴坐标，当然你也可以互换；了解了结果类型后就可以人工干预算法计算的布局，稍作调整；第 4 行代码使用 plot 函数绘制网络图，看一看算法计算的布局，如果不满意可以调整。

第 5 行代码中 V(weaponnet)\$sg == 1 筛选第 1 小组的 X 轴坐标，并向右调整 100 个单位 ( 是不

是用到了中学函数位置调整的知识)；第 6 行代码筛选 1 小组的 Y 轴坐标，并向下调整 100 个单位；第 7 行代码使用 plot 函数绘制调整后的网络图，可以看到 1 小组的节点集体向下移动了；第 8 行和第 9 行代码调整单个点的坐标，通过查看 V(weaponnet)，我们知道印度的节点编号为 79，那么就很容易按编号调整了。

效果如图 9-5 所示

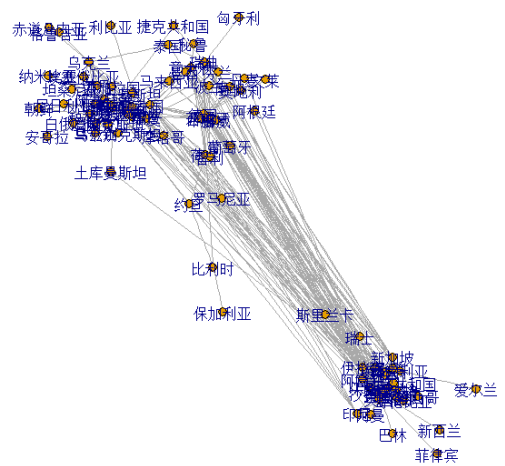


图 9-5

### 9.3.3 全球某货物交易圈：寻找各自的小伙伴

上面我们清洗整合了数据，对数据的汇总进行了调整，同时对节点分组并计算节点的重要性，但是这些只能算作分析思路，具体结果怎样我们并不知道，本节我们就需要对结果可视化并进行分析，所以分析工作是一个流程，而不只是建模一个环节。

#### 1. 全球某货物交易圈和他们的小伙伴

首先根据以上计算结果，绘制全球某货物交易圈，并对它们进行分组，同时还要展示出各国的重要性及各国交易关系的强弱，如下所示。

- 按颜色分组

```
1 require("igraph")
2 colors <- c(rgb(0, 130, 137, 50, max = 255), rgb(252, 97, 107, 50, max = 255), rgb(186, 9, 241, 50, max = 255))
3 V(weaponnet)$color <- colors[V(weaponnet)$sg]
4 temp <- get.edgelist(weaponnet)[,2]
5 E(weaponnet)$color <- V(weaponnet)[temp]$color #为 edge 的颜色赋值
6 colors <- c(rgb(0, 130, 137, 180, max = 255), rgb(252, 97, 107, 180, max = 255), rgb(186, 9, 241, 180, max = 255))
```

```
= 255), rgb(186, 9, 241, 180, max = 255))
  7 V(weaponnet)$color <- colors[V(weaponnet)$sg]
```

按颜色分组要做到两点：首先对节点分组，即不同节点组的颜色不同；其次，需要对边线分组，即不同组的边线颜色不同。前者很容易做到，上述代码中，第2行代码生成三组特定的颜色和透明度备用；第3行按节点组编号提取颜色，产生一个和节点等长且顺序对应的颜色值向量，同时赋值于节点的颜色属性，这样节点就具有了颜色；然后使用节点颜色给边线颜色赋值，第4行代码使用 `get.edgelist` 可以获得边线（关系），关系是双边的，所以 `get.edgelist` 函数获取的是一个2列矩阵，提取边线的第2列作为确定边线颜色的节点，即第2列的节点具有什么样的颜色，这条边线就被赋予什么样的颜色属性（你也可以选择第1列）；第5行代码中 `V(weaponnet)[temp]$color` 根据 `temp` 筛选网络节点，并提取节点的颜色属性赋值给相应边线的新属性：`color`；设定边线的颜色后，需要对节点的透明度进行调整，因为如果透明度一样，边线会影响节点的颜色效果，第6行代码将三种颜色的透明度都调整为180；第7行代码将调整后的颜色按照节点编号筛选重新赋值给节点的颜色属性，这样分组就完成了，接下来可以绘图了。如下所示。

- 某货物交易网络图

```
1 require("extrafont") # font_import()#引入字体
2 loadfonts(device="win")
3 png(filename = "全球某货物交易.png", width = 1200, height = 1200)
4 par(mai=c(0,0,0,0))
5 plot(weaponnet, layout = layout, vertex.size = log10(V(weaponnet)$size*
10000)*3, vertex.color = V(weaponnet)$color, edge.color = E(weaponnet)$color,
edge.width = E(weaponnet)$weight/1500, edge.arrow.mode = "-", vertex.label.dist
= 0.4, vertex.label.cex = 1.5, vertex.label.font = 2, vertex.label.family =
"Microsoft YaHei", vertex.label.color = rgb(49, 44, 13, 150, max = 255))
6 dev.off()
```

上述代码中，前两行代码用于引入 Windows 的字体（第3章已经讲过）；第3行代码开启 PNG 绘图设备，将图片保存为当前目录下的“全球某货物交易.png”文件，同时设定文件的宽度和高度；第4行代码通过 `par` 函数设置图片边缘宽度；`plot` 函数用于绘图，`layout` 指定布局（经过人工调整），`vertex.size` 设置顶点大小，之前我们将 PageRank 计算的节点重要性赋值给了节点的 `size` 属性，现在将 `size` 属性乘以10000后做对数转化，然后结果再乘以3作为节点的大小（这些数字根据个人喜好设定），设置办法需要人工尝试调整，不然数据点就会出现过小或过大的可能，失去了图片美观性，`vertex.color`、`edge.color` 分别用于指定节点和边线颜色，边线的 `weight` 属性是两国之间的交易额，除以1500后赋值用来映射边线宽度 `edge.width`，`edge.arrow.mode` 设定边线类型，`vertex.label.dist` 节点标签到节点的距离，避免标签和数据点重合，`vertex.label.cex` 设置节点标签字体大小，`vertex.label.font` 设置节点标签的字体类型，有粗体、斜体、粗斜体可选，`vertex.label.family` 指定标签字体使用 Windows 下的微软雅黑字体；最后一行代码关闭 PNG 设备。

这样去当前目录下就可以找到绘图结果了，从绘图结果可以看出，全球可以分为三个小组，分别以美国、俄罗斯、德国为首，令人震惊的是俄罗斯除了有中国这个好伙伴，还有印度，而且其交

易规模不亚于中国。另外一个特征是曾经的两极群体仍然保持一定的极性，俄罗斯和中国、印度的交易占据其某货物交易的绝大部分，美国也同样，和英国、韩国、日本的交易非常大，但是再回观欧洲，德国和其他国家的关系都是细若游丝的弱关系，但仍然能帮助德国成为欧洲的“老大”。

另外，通过查看 PageRank（网页排名），发现欧洲的领导者德国竟然比印度和俄罗斯的重要性低很多，我们能得出中国在全球某货物交易网络中的实力已经超过了德国、英国吗？尽管这重要性反映的是总交易，但是可能有些国家以技术进口为主呢，这种交易额怎么能和以技术出口为主的国家比呢？所以如果不知道来龙去脉不要轻易下结论。

## 2. 用金字塔图分析，为什么 PageRank 失效了

不如看看各国出口的情况和它们的网页排名的关系，看看它们之间的关联程度有多大？首先要对数据进行调整，计算各国的出口总额，然后将出口总额和 PageRank 值捆绑在一起绘图分析结果。

### • PageRank 与出口调整

```
1 countryex <- aggregate(dollar_m ~ export, exvsim, sum)
2 countryex$dollar_m <- countryex$dollar_m/100
3 countryex$variable <- rep("出口总额", length(countryex[, 1]))
4 names(countryex) <- c("country", "value", "variable")
5 countryex <- countryex[countryex$value > 0.5, ]
6 pagevalue <- data.frame(pagesize)
7 pagevalue$country <- row.names(pagevalue)
8 pagevalue <- pagevalue[pagevalue$country %in% countryex$country, ]
9 pagevalue$variable <- rep("pagerank 值", length(pagevalue[, 1]))
10 names(pagevalue) <- c("value", "country", "variable")
11 pagevalue$value <- pagevalue$value*100
12 countryex <- countryex[order(countryex$value), ]
13 countryex <- rbind(countryex, pagevalue)
```

上述代码中，第 1 行代码将上面保存的 exvsim 数据框按照出口国透视汇总；第 2 行代码将交易额单位由百万美元调整为亿美元；第 3 行代码将字符“出口总额”复制和 countryex 等长的次数后赋值给数据框的新变量 variable；第 4 行代码重命名各列；第 5 行代码筛选销售总额大于 5 千万美元的国家；第 6 行代码将上面的 PageRank 结果转化为数据框，数据框的行名称就是相应的国家名；第 7 行代码 row.names 函数获取行名称并赋值给新变量 country（国家）；第 8 行代码筛选在出口国家汇总表出现过的国家的 pagerank 数据；第 9 行代码将字符“pagerank 值”复制为和数据框等长的向量并赋值给新变量 variable；第 10 行代码重新命名数据框；第 11 行代码将 PageRank 值放大 100 倍；第 12 行代码 rbind 将两个数据框按行捆绑在一起，head 一下两个数据框，不难看出，刚开始它们的列顺序是不一样的，但 rbind 函数自动进行了匹配，这也是 rbind 速度慢的原因之一。

### • 自编函数、辅助数据

```
1 add_credits = function(fontsize = 12) {
2   grid.text("大音如霜工作室", x = 0.99, y = 0.02, just = "right", gp =
gpar(fontsize = fontsize,
3     col = "#777777", fontfamily = "STXingkai"))
```



```

4 }
5 title_with_subtitle = function(title, subtitle = "") {
6   ggtitle(bquote(atop(. (title), atop(. (subtitle)))))
7 }
8 library(grid)
9 library(extrafont)
# font_import()#引入字体
10 loadfonts(device = "win")

```

- 载入主题

```

1 theme1 <- theme(panel.background = element_rect(fill = rgb(red = 242,
2   green = 242, blue = 242, max = 255)),
3   plot.background = element_rect(fill = rgb(red = 242,
4   green = 242, blue = 242, max = 255)),
5   plot.title = element_text(size = rel(1.2), family =
"STXingkai", face = "bold", hjust = 0.5, colour = "#3B3B3B"),
6   panel.grid.major = element_line(colour=rgb(red = 146, green
= 146, blue = 146, max = 255),size=.75),
7   panel.border = element_rect(colour = rgb(red = 242,
8   green = 242, blue = 242, max = 255)),
9   axis.ticks = element_blank(),
10  axis.text.x = element_text(colour = "grey20", size = 8),
11  axis.text.y = element_text(colour = "grey20", size = 10),
12  axis.title.y = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = 0.5),
13  axis.title.x = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = -0.5),
14  legend.background = element_rect(fill = rgb(red = 242,green
= 242, blue = 242, max = 255)),
15  legend.position = "bottom")

```

以上已经在第3章讲得非常清楚，代码未作改变。

- 绘制金字塔图

```

1 library(ggplot2)
2 countryex$country <- as.character(countryex$country)
3 temp <- countryex[countryex$variable == "出口总额", ]
4 temp <- temp[order(temp$value),]
5 countryex$country <- factor(countryex$country, levels = temp$country)
6 p <- ggplot(data = countryex, aes(x = country, y = value, fill = variable)) +
7   geom_bar(data = subset(countryex, variable == "出口总额"), stat = "identity") +
8   geom_bar(data = subset(countryex, variable == "pagerank 值"), stat =
"identity", position = "identity", mapping = aes(y = -value)) +
9   xlab("") +
10  ylab("") +
11  scale_y_continuous(labels = abs) +

```

```

12 coord_flip() +
13 scale_fill_manual(values = c("#2c7fb8", "#FF1493"), name = "图例") +
14 annotate("text", x = 3, y = 60, label = "单位: 亿美元", size = 4, fontface
= "bold") +
15 title_with_subtitle("全球某货物交易出口国排行") +
16 theme_bw(18) +
17 theme1

```

前面已经说过在使用 ggplot 绘图时，会将字符变量当作因子型，其顺序不是按照因子型变量对应的数值排序的，而是按照因子水平排序的，因此我们需要调整 country 变量的因子水平，以实现条形图或柱形图按照数值从大到小或从小到大排列，上述第 2 行代码将变量转化为字符型；第 3 行代码提取 variable == "出口总额"的数据赋值给 temp，第 4 行代码对 temp 按照 value 排序，temp 里的 value 只是出口总额，也就是说对 temp 按照出口总额排序（默认升序）；第 5 行代码对 countryex 里的 country 变量重新编因子水平，即变量不变，因子水平使用 temp 里的 country 列里面，各国出现的先后顺序设定，这样 countryex 里的 country 就按照交易额从小到大排序了，以上过程在第 3 章已经反复使用过；第 6 行代码开始使用 ggplot 绘图，将 country 映射为 X 轴，将 value 映射为 Y 轴，variable 映射为填充色；第 7 行代码绘制条形图图层，这里 subset 函数进行了数据筛选，仅绘制“出口总额”的数据；第 8 行代码筛选“pagerank 值”绘制条形图，这里使用了 mapping 参数将 Y 轴映射为 value 的负值，如果你仅仅执行到这一步，得到的图如图 9-6 所示。

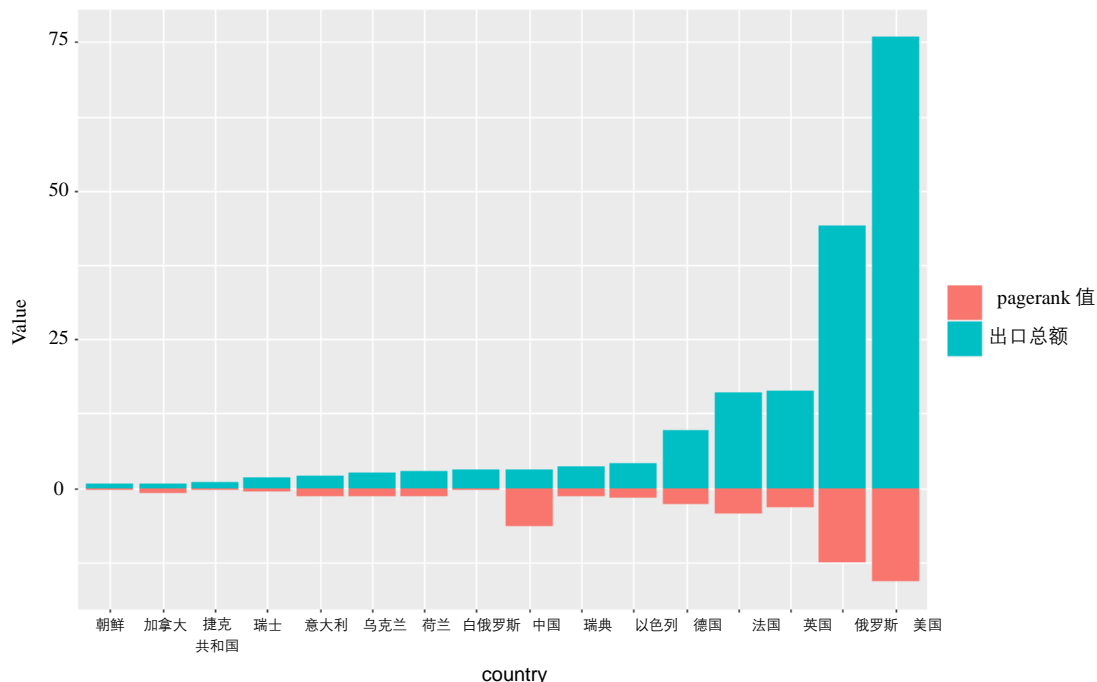


图 9-6

ylab 和 xlab 分别用于设定 Y 轴和 X 轴名称，这里将其设置为空，scale\_y\_continuous 用于设定 Y 轴标签，调用了 abs 函数对 Y 轴映射值取绝对值，这样标签就变成了正值；coord\_flip 函数将 X、Y 轴旋转一下；scale\_fill\_manual 用于设置填充色系；annotate 给图片添加必要的计量单位说明；自编函数 title\_with\_subtitle 添加主标题；最后两行代码设置主题，绘图的内容绝大多数已经在第 3 章讲过了。

- 保存图片

```
png(filename = "pagerankvsexport.png", width = 500, height = 500)
print(p)
dev.off()
```

启动 PNG 设备将图片保存，从图 9-7 中我们可以看出，中国的出口总额远远低于美国、俄罗斯等，PageRank 值很高的印度甚至没有进入出口排行榜，所以这里的 PageRank 值不能代表国家的水平，仅仅反映一个国家在网络中的重要性，此重要性既包括出口关系又包括进口关系，印度、中国还是处于网络的底端，花钱购买，其重要性就是如果没有它们，这一行业会损失不小的市场；产生这种错觉主要是因为上面 PageRank 是基于非指向型网络的，各国在指向型网络中的重要性也许是另一番面目，感兴趣的读者可以构建一个指向型网络，然后使用 PageRank 计算一下各国的重要性，相互印证一下。

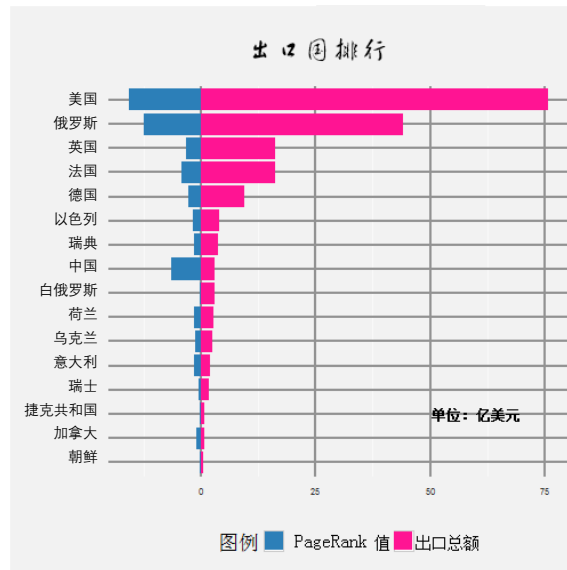


图 9-7

## 9.4 中国电影演艺圈到底有没有“圈”

演艺圈人士常常登上各大媒体的头条，也是人们茶余饭后的主要谈资，艺人的花边八卦各领一时风骚。无论是艺人还是观众可能对演艺圈的结构都是盲人摸象，很少有人说得明白。

在 8.4 节我们抓取了大量的电影数据，其中就有每一部电影的演员，假设两个演员同时出演一部电影，可以认为他们的关系还不错，如果他们经常“合作”，那我们就认为俩人的关系很好，这样就可以根据数据构筑一个演艺圈的关系网络图，进而分析明星的合作关系。

在回答关于演艺圈的诸多问题之前，首先要看看我们拿到的是什么样的数据。说到这里，可能很多数据挖掘工程师都遇到过这样的情景：一群产品经理坐在一起进行“头脑风暴”，突然一个想法灵光闪现，大家纷纷添油加醋，汇报给领导，领导说不错，你们去做吧，尽快出产品。好吧，回来抽取数据稍作探索就会发现各种垃圾需要清洗，需要整合，更无情的是竟然无法有效关联（各表之间仅关联一小部分），甚至大量缺失及反直觉的问题也可能出现。怎么办，一路妥协，最后把产品做成了“垃圾”（shit）。根据经验，探索数据是必要的，不要以为你知道表格的主键就可以了，主键可能只关联一丢丢，不深入地探索数据就架构数据产品，那就是空中盖楼房，搬起石头砸自己的脚。更可笑的是，很多产品的问题和矛盾竟然是由客户提出的，数据挖掘工程师并没有发现。

### 9.4.1 数据清洗与整形

只有“财大气粗”的公司才会有专门的数据清洗人员，每一个数据挖掘工程师都要有获取数据、清洗整理数据、分析挖掘数据的能力。在我工作的大多数时间里，除了某些教科书里的数据，基本上都要花费时间去清洗整理。不要怕，做得越多越熟练，做得越多对数据越了解。下面看看我们拿到了什么样的原始数据。

- 看看数据有多烂

```
1 filmdetail <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的  
社交网络关系/data/filmdetail.csv", header=TRUE, stringsAsFactors = F)  
2 head(filmdetail)
```

根据我多年“行走江湖”的经验判断，这份数据还算不错，经过简单的整理就可以应对本章节的问题。首先电影的演员（actor）、导演（directedBy）、编剧（writer）、地区（region）这些列整理得非常整齐，不需要我们从资料列里解析出来，如果你想锻炼自己的水平可以想办法从资料中分离出来这些信息。

首先我们要从数据中划出一个范围，比如什么样的电影才算是中国演艺圈的，尽管有些演员会客串好莱坞电影，但如果电影不是在国内上映，也不算中国演艺圈，这里仅将上映地区为中国大陆、中国港澳台的电影算在中国演艺圈内。

- 圈定中国演艺圈

```
1 unique(filmdetail$region)  
2 filmdetail$region <- gsub(" ", "", filmdetail$region)
```

```

3 unique(filmdetail$region)
# 中国大陆|中国|中国香港|中国台湾
4 filmdetail <- filmdetail[grepl("中国大陆|中国|中国香港|中国台湾",filmdetail$region),]
5 temp <- strsplit(as.character(filmdetail[, "dat"]), "-")
6 temp <- unlist(lapply(temp, function(x) x[1]))
7 filmdetail$dat <- as.numeric(temp)
8 filmdetail <- filmdetail[filmdetail$dat > 2009,]

```

中国大陆、中国港澳台的写法不同，原始数据书写不规范，不如看看总共写了哪些地区，上述第1行代码对地区去重后发现竟然有185个地区；仔细观察就会发现“中国大陆”前后出现了多次，但是我们已经去重了，说明“中国大陆”前后可能存在空格；第2行代码对地区使用 `gsub` 函数查找空格并替换为空；第3行代码再次去重查看，只剩下124个地区；但还是可以看到几种不同的写法，这里使用 `grepl` 函数在 `filmdetail$region` 里查找任何一个和中国有关的词汇，“|”表示“或”的关系，表示查找“中国大陆”或“中国台湾”，`grepl` 函数名中的“l”表示函数返回的结果是逻辑值（logic），如果找到了就返回真，找不到就返回假，根据返回值就可以筛出所有符合定义的中国电影了，但也有些例外，比如筛出了一些明显是欧美或日韩的电影，但是上映地区却是中国大陆，不过它们会在后面的筛选中被筛掉。

第5行代码将 `dat` 列的日期按照“-”分裂，分裂后形成一个 `list`，第1个元素是年份；第6行代码使用 `lapply` 将所有的第1个元素提取出来，并使用 `unlist` 函数将其解散为 `vector`，这样我们就提取了年份；第7行代码使用 `as.numeric` 函数将字符转化为数字，然后赋值给 `dat` 列，就将原来比较混乱的日期替换为年份了；最后一行代码筛选2010年至今的电影。

我们还要划定一个艺人的范围，这里仅将演员作为演艺圈的艺人，不包含导演和编剧（范围可以自己设定），这样只需要提取 `movieid`、`actor` 列，稍作整形就可以作为网络分析的数据了，如下所示。

#### • 数据整形

```

1 filmstar <- filmdetail[,c("movieid", "actor")]
2 filmstar <- unique(filmstar)
3 temp <- table(filmstar$actor)
4 temp <- temp[temp < 2]
5 temp <- names(temp)
6 filmstar <- filmstar[!(filmstar$actor %in% temp), ]
7 filmstar <- filmstar[!is.na(filmstar$actor), ]
8 filmstar <- split(filmstar$actor, filmstar$movieid)
9 temp <- lapply(filmstar, length)
10 filmstar <- filmstar[temp > 1]
11 temp <- lapply(filmstar, function(x){
12   a <- t(combn(unlist(x), 2))
13 })
14 filmnet <- do.call("rbind", temp)
15 filmnet <- data.frame(filmnet, stringsAsFactors = F)

```

上述代码中，第 1 行代码提取包含 actor（演员）的列；第 2 行代码对 filmstar（明星）去重；第 3 行代码对 actor（演员）列计数，可以算出每一个演员这些年参演了多少部电影；第 4 行代码筛出那些仅演过 1 部电影的演员，这个数值可选（本文中选择将其注释掉或者设置为 1，即筛出 1 部电影都没演过的演员，是的，不存在这类人，主要是这里设定了一个可以调节的参数），但我认为一个演员 5 年才出一部电影也比较不靠谱；第 5 行代码获得这些演员的名字（这里不再做进一步清洗）；第 6 行代码筛除上一步获得的那些演员；第 7 行代码筛除演员为空的电影；第 8 行代码使用 split 函数将 filmstar\$actor 按照 filmstar\$movieid 切分为 list，list 的每一个元素是一个小 data.frame；当然有些电影里的演员少于两个，无法构成一次关系，所以第 9 行代码使用 length 计算每一部电影里的演员数；第 10 行代码提取演员数大于 1 的电影；做成 edgelist 需要将每一部电影里的演员进行一次两两组合，combn 函数可将一个向量中的元素按照指定的元素数目组合，可以查看帮助文档；第 11 行代码将 filmstar 每一部电影应用于匿名函数，combn 完成组合，t 函数将 combn 的结果进行行列转置；最后两行代码将组合后的结果（list）转化为数据框，这样输入 igraph 的数据结构基本上就整理结束了。

但是还有一种情况就是使用 graph.edgelist 对象时，如果我们需要对边线（edge）加权，就需要计算出每一对演员之间的合作次数，但是会出现 a、b 和 b、a 的现象，即镜像重复，之前已经有一套解决方案，但是非常笨拙，这里我们测试一种新方法。

比如说一个数据框 x 中第 1 行的数值为 1、2，与第 2 行的数值 2、1 为镜像重复，要改变这种状态，只需要将任意一行的两列数中较小的放在第 1 列，较大的放在第 2 列即可，这样处理比较高效而且代码简洁。如下所示。

- 专治镜像重复（1）

```
1 x <- data.frame(m = 1:2, n = 2:1)
2 x <- transform(x, T1 = ifelse(m <= n, m, n))
3 x <- transform(x, T2 = ifelse(m > n, m, n))
4 x
#   m n T1 T2
# 1 1 2  1  2
# 2 2 1  1  2
# 字符比较
5 x <- data.frame(m = c("孙", "李"), n = c("李", "孙"), stringsAsFactors = F)
6 x <- transform(x, T1 = ifelse(m <= n, m, n))
7 x <- transform(x, T2 = ifelse(m > n, m, n))
8 x
```

上述代码中，第 1 行代码创建一个镜像重复数据框 x；第 2 行代码使用 transform 函数给 x 新加一列 T1，T1 的内容使用 ifelse 函数判断填充，如果 x 的 m 列小于等于 n 列，则将 m 列的数据赋值给 T1，否则将 n 列赋值给 T1，这样 T1 列就是 m 列和 n 列数值较小的数据；同理第 3 行代码添加一新列 T2，如果 x 的 m 列大于 n 列，则将 m 列的数据赋值给 T2，否则将 n 列赋值给 T2，这样 T2 列就是 m 列和 n 列数值较大的数据，这样我们就将 m、n 列的镜像重复给翻转成了 T1 和 T2 列的样子

了；其实字符也可以比较大小，比较的是字符的编码大小，最后几行代码用于演示调整中文的镜像重复。下面的方法代码比较冗长，但是可以设定是保留 ab 模式还是保留 ba 模式，仅供参考。

- 专治镜像重复 (2)

```
1 x <- data.frame(m = c("m", "b", "c", "a"), n = c("b", "a", "b", "b"), l = c(1,
1, 1, 1), stringsAsFactors = F)
2 temp <- unique(c(x$m, x$n))
3 x$temp <- paste(x$m, x$n, sep = "")
4 temp <- t(combn(unlist(temp), 2))
5 temp <- paste(temp[,1], temp[,2], sep = "")
6 datatemp <- x[x$temp %in% temp,]
7 datatemp <- datatemp[, c("n", "m", "l")]
8 names(datatemp) <- c("m", "n", "l")
9 data <- x[!x$temp %in% temp,]
10 data <- data[, c("m", "n", "l")]
11 data <- rbind(data, datatemp)
```

统计两个演员的合作次数，首先需要处理镜像关系，然后使用透视表透视统计次数即可。如下所示。

- 统计合作次数

```
1 filmnet$logic <- rep(1, length(filmnet[,1]))
2 filmnet <- transform(filmnet, T1 = ifelse(X1 <= X2, X1, X2))
3 filmnet <- transform(filmnet, T2 = ifelse(X1 > X2, X1, X2))
4 filmnet <- aggregate(logic ~ T1 + T2, filmnet, sum)
5 names(filmnet) <- c("T1", "T2", "weight")
```

上述代码中，第 1 行代码添加统计次数的辅助列 logic，所有值都为 1；第 2 行和第 3 行代码用于处理 ab 和 ba 的镜像重复；第 4 行代码使用 aggregate 函数按照 T1、T2 两列对 logic 分组求和；第 5 行代码重命名，这样 filmnet 就是全国每对演员的合作次数清单表了。

## 9.4.2 看看演艺圈长什么样

演艺圈长什么？我并没见过可视化的证据。实际上，人们定义的演艺圈不是根据网络关系定义的，它是一条职业分解线，比如你的职业和演艺圈相关你就是演艺的人，不相关你就是圈外的人，这样定义，很多群众演员也是演艺圈的人，但是如果根据网络关系判断，可能又是另一番景象。此次分析虽然不能完全代表演艺圈的状况，但也能旁敲侧击了解大概。

```
1 rm(temp, filmdetail)
2 gc()
3 library(igraph)
4 starnet <- graph.edgelist(as.matrix(filmnet[, 1:2]), directed = F)
5 E(starnet)$weight <- filmnet[, 3]
6 library("ForceAtlas2")
7 set.seed(1639)
```

```
8 layout <- layout.forceatlas2(starnet, iterations = 5000, plotstep = 500)
9 png(filename = ("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 9 章不可不说的社交网  
络关系/plot/starnet1.png", width = 600, height = 600)
10 plot(starnet, layout = layout, vertex.size = 3, vertex.color = rgb(0, 137,  
130, 180, max = 255), vertex.frame.color = NA, vertex.label = NA)
11 dev.off()
12 starnet
13 graph.density(starnet)
14 save(list = ls(), file = "H:/starnet.RData")
```

上述代码中，第 1 行代码移除一些暂时无用的对象；第 2 行代码收集内存，这两步为了尽快腾出内存空间；第 4 行代码构建 igraph 对象；第 5 行代码将合作次数赋值给边线的权重属性；因为生成布局是一个随机过程，为了保持你和我的结果一致，第 7 行代码设置随机种子；第 8 行代码生成网络图的布局，你也可以使用比较常见的布局生成方式，通过“?layout”可以查看很多种生成布局的算法；第 9 行代码启动 PNG 设备；第 10 行代码绘制演艺圈的网络，其中的参数在 9.3 节已经基本都熟悉了；最后两行代码输出一些基础评价指标，用于评价一下演艺圈的合作关系。

在这段时间里，2000 多名影视演员进行了 24000 多人次的合作。演艺圈的合作网络密度仅为 0.01，没落的人人网，其社交网络密度平均不足 0.03，可见演艺圈的合作网络密度多么低下。因此反映出演艺圈的合作比较松散，当然演艺圈有它固有的性质决定了圈子的密度，这些性质致使大部分演员被隔离在圈子的外层，令人忧伤的是很多演员演了戏，却始终没有走进演艺圈（见图 9-8）。这种现象也比较符合二八定律，任何一个行业，20% 的人攫取了行业的 80% 福利，而另外 80% 的人仅仅分得一些残羹冷炙。但根据我这么多年分析大数据的经验，其实二八定律已经在很多行业接近三七分了，也就是说特殊人群和普通人群的差异正在缩小。

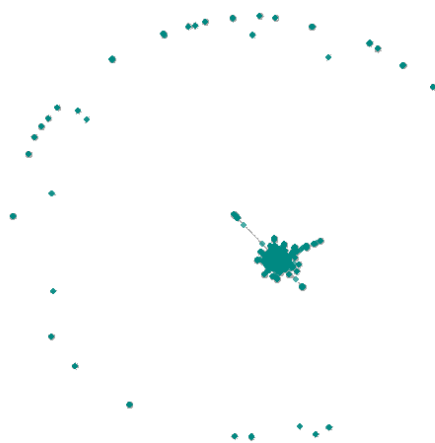


图 9-8

其实这张图可以做得更好看，限于篇幅，不再赘述，可参看电子版代码或者加群讨论。



### 9.4.3 谁才是演艺圈的“关系户”

那么谁才是演艺圈的核心？恐怕抛出来这个问题，又将掀起一场“口水大战”，轻易给出答案的人注定会淹没在口水的海洋中。这里不谈演技、不谈粉丝数量等，简单就最近五六年的合作关系而言，关系比较多的当属刘德华、林雪、吴彦祖、曾志伟、廖凡，也就是演艺圈的“五大关系户”，出人意料的是林雪这个“龙套王”，几乎每部杜琪峰的电影里都有他的身影，如果你看片足够仔细，则会发现身影还不止一个。

- 添加约束条件（1）

```
1 actor <- data.frame(names(degree(starnet)), unlist(degree(starnet)))
2 names(actor) <- c("actor", "edges")
3 actor <- actor[order(actor$edges, decreasing = TRUE),]
4 head(actor, 10)
#      actor edges
# 930  刘德华  197
# 908   林雪  185
# 1600 吴彦祖  183
# 85   曾志伟  180
# 873   廖凡  170
# 1691 徐峥  163
# 1926 张涵予 158
# 313  冯绍峰 157
# 730   李晨 155
# 416   郭涛 153
5 summary(actor$edges)
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    2.0     9.0    15.0    22.6   25.0   197.0
```

上述代码中，第1行代码生成一个网络的所有顶点，以及这些顶点所发出的关系数，degree 获得网络中任一节点连接的节点数，names 函数获得相关节点的名称，和 degree 一起组成了一个一一对应的数据框；第2行代码对数据框的列重新命名；第3行代码使用 order 将数据框按照 edges 列的降序排列；第4行代码查看 TOP10，刘德华位居第一；summary 函数看一下统计摘要，发现有一半演员的伙伴关系低于15人，大多数演员的关系低于平均值（见第5行代码）。

如果我们将合作人次大于等于15设为一个界限，则可以帮助我们清除掉一些“花边”。不妨对去掉“花边”的娱乐圈再进行一次可视化，看看发生了哪些变化。如下所示。

- 添加约束条件（2）

```
1 temp <- V(starnet)[degree(starnet) < 60]
2 starnet <- delete_vertices(starnet, temp)
3 V(starnet)$label <- NA
4 temp <- degree(starnet) > 80
5 V(starnet)[temp]$label <- V(starnet)[temp]$name
6 V(starnet)$size <- 3
```

```

7 V(starnet)[temp]$size <- 2*log(degree(starnet)[temp])
8 V(starnet)$color <- rgb(0, 137, 130, 100, max = 255)
9 V(starnet)[temp]$color <- rgb(253, 141, 149, 100, max = 255)
10 layout <- layout.forceatlas2(starnet, iterations = 150000, plotstep = 500)
11 png(filename = "starnet2.png", width = 1200, height = 1200)
12 plot(starnet, layout = layout, vertex.size = V(starnet)$size, vertex.
color = V(starnet)$color, vertex.frame.color = NA, vertex.label = V(starnet)
$label, edge.width = E(starnet)$weight/100)
13 dev.off()

```

上述代码中，第 1 行代码提取关系数少于特定值“60”的节点；第 2 行代码使用 `delete_vertices` 函数删除这类节点；第 3 行代码添加新属性 `label`，并全部赋值为空；第 4 行代码判断任一节点的 `degree` 是否大于 100；第 5 行代码将 `degree` 大于 80 的节点筛选出来，并将 `label` 赋值为相应的节点名称；第 6 行代码添加新属性 `size`，全部赋值为 3；第 7 行代码将 `degree` 大于 80 的节点的 `size` 赋值为相应的 `degree` 值，赋值之前进行了一次 `log` 转化，免得差异太大（很多统计变换是为了扩大和缩小视觉差异，所以读图时千万要看清）；第 8 行代码为节点添加新属性 `color`，并统一设置颜色和透明度；第 9 行代码设置 `degree` 大于 80 的节点颜色属性；下面的是绘图，不再详述。

其实进入核心圈以后，关系数量和参演的电影数量正相关。每个演员对待作品可能存在很大的差异，有些人追求质量，有些人追求数量。对于演员，同一段时间的价值可能比普通人更为重要，6 年时间对普通人而言不长不短，可是对于一个演员就意义非凡了，6 年后的演艺圈可能就是另一番天地。抓住黄金时间尽可能多地出演质量较好的作品，对于提高演员的声望、地位、演技和收入尤其重要（以拍广告谋生者除外）。实际上也有一些名不见经传的艺人能通过 6 年时间斩获无数演技大奖。

对节点的中介作用有一个经典的刻画叫作“中间度”。中间度衡量了节点作为中介的程度，当网络中某个个体的中间度较大时，我们认为它在很大程度上起到了中介和沟通的作用，前面我们已经讲过。如下所示。

- 娱乐圈的中介人

```

1 starnet <- graph.edgelist(as.matrix(filmnet[, 1:2]), directed = F)
2 E(starnet)$weight <- filmnet[, 3]
3 V(starnet)$betweenness <- betweenness(starnet, directed = F)
4 actorbetweenness <- data.frame(names(V(starnet)), V(starnet)$betweenness)
5 names(actorbetweenness) <- c("actor", "betweenness")
6 actorbetweenness <- actorbetweenness[order(actorbetweenness$betweenness),]
7 tail(actorbetweenness, 10)
8 summary(actorbetweenness$betweenness)
#      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#      0.00   0.00   2.05 1861.00 455.90 80210.00

```

上述代码中，第 1 行和第 2 行代码重新构建网络；第 3 行代码为网络添加新属性 `betweenness`，使用 `betweenness` 函数计算网络中每个节点的中间度；第 4 行代码提取节点名称、中间度组合为数据框；第 5 行代码重命名数据框；第 6 行代码按照 `betweenness` 从小到大排序；第 7 行代码查看中介度最高的前十名；第 8 行代码输出中介度的统计摘要，很明显是一个偏分布，绝大多数艺人的中介度

低于平均水平，范冰冰夺得中介度魁首。我一直认为国内有两个营销团队很厉害：一个是加多宝，一个是范冰冰团队。

推荐系统有两种，一种是基于产品的推荐系统，一种基于用户的推荐系统，所谓基于产品的推荐是指当我们给用户推荐产品时，通过计算产品之间的相似性推荐给用户；所谓基于用户的推荐系统，即当我们给客户推荐产品时，暂不考虑产品之间的相似性，而是以用户之间的相似性进行推荐，比如用户 a 和用户 b 的行为很相似，那么我们就可以将 b 购买的商品推荐给 a。

#### 9.4.4 用 Apriori 算法查查演艺圈合作的“朋友”关系

Apriori 算法可以构建一个推荐系统，仔细思考，会发现 Apriori 算法是基于用户的推荐系统，算法不考虑啤酒和尿布本身有什么相似性( 我不相信这个故事 )，而是根据用户的选择行为进行了推荐，这里我们用 Apriori 算法找出一些支持度比较高的组合，就可以做一些简单的推荐了，比如导演选择了某演员，是不是也会选择该演员的好朋友呢？

- 使用 Apriori 算法挖掘好朋友

```
1 library("arules")
2 trans <- as(filmstar, "transactions")
3 trans.apriori <- apriori(trans, parameter = list(support = 0.01, confidence = 0.7, maxlen = 2))
4 inspect(trans.apriori)
```

Apriori 算法来源于 arules 包，apriori 函数需要将数据转化为 transactions 对象，as 函数可以将数据转化特定的对象类型，上述第 2 行代码将 filmstar 转化为 transactions 对象，filmstar 是一个 list，其每一个元素的名字是电影的 id ( 可以理解为订单 id )，里面包含的是参演电影的演员 ( 可以理解为每一个订单购买的商品 )，第 2 个参数是最终要转化为的对象，其实也可以将数据框转化为 transactions，只是要求数据框要包括两列：第 1 列电影 id，第 2 列电影演员名称，而且两列数据类型都必须是因子；第 3 行代码使用 apriori 寻找符合要求的规则，第 1 个参数指定 transactions 对象，第 2 个参数用于指定寻找什么样的规则，这里我们要求支持度不少于 0.01，置信度不少于 0.7 的规则，同时要求规则包括的元素数不大于 2，至于什么是支持度可以参看前面专讲算法章节的通俗说法；第 4 行代码使用 inspect 查看找到的规则，可以看到吴彦祖和方中信两人的支持度、置信度都比较高。有人说，规则的置信度一般要求大于 0.75 才有意义，是不是这样大家可以设计方案研究一下，但是可以直白点说，我觉得啤酒和尿布的故事实在比较扯。

- 支持度 ( support ) :  $P(A \cup B)$ ，即 A 和 B 这两个项集在事务集 D 同时出现的概率。
- 置信度 ( confidence ) :  $P(B | A)$ ，即在出现项集 A 的事务集 I 中，项集 B 也同时出现的概率。

trans.apriori 的数据类型是 rules，不是我们熟悉的类型，可以将之转化为数据框，通过下面的代码可以将 rules 转化为数据框。

- 将 Apriori 结果转化为数据框

```
1 class(trans.apriori)
2 actorapriori <- data.frame(lhs = labels(lhs(trans.apriori))$elements, rhs
```

```
= labels(rhs(trans.apriori))$elements, trans.apriori@quality)
3 actorapriori$lhs <- gsub("\\{|\\}", "", actorapriori$lhs)
4 actorapriori$rhs <- gsub("\\{|\\}", "", actorapriori$rhs)
```

上述代码中，第 1 行代码查看对象的数据类型；第 2 行代码将 rules 转化为数据框，一条规则“{方中信} => {吴彦祖}”分为左右两个部分，lhs（left-hand-sides）提取 rules 的左半边演员，labels(lhs(trans.apriori))\$elements 提取左边演员名称，labels(rhs(trans.apriori))\$elements 提取右边演员名称，trans.apriori@quality 提取规则的一些指标，这样就转化为数据框了；但是演员名称的外面包括了一个大括号，当然需要把它除掉，“\\{|\\}”的模式（pattern）表示查找“{”或“}”，“|”表示或的关系，“\\”表示逃逸符，因为大括号在正则表达式中具有特殊含义，逃逸符就是表示这里的大括号不是那个特殊含义，真的是为了匹配大括号，gsub 函数查找符合指定模式的部分然后替换为另一种模式，这里我们替换为空，到这里才真正将 rules 对象转化为了数据框。

## 9.4.5 给范冰冰推荐合作伙伴

除了 Apriori 之外，我们简单介绍另外一种推荐方法，通过优势排名进行推荐。不如给范冰冰推荐几个合作伙伴，给范冰冰推荐合作伙伴时要注意符合几个原则：（1）推荐对象在最近 6 年内非范冰冰合作者（应该这样）；（2）推荐对象与范冰冰的共同合作者较多，方便通过熟人接触；（3）推荐对象在合作网络中中介度较高；（4）推荐对象的合作关系比较多。根据以上原则使用均权算法为范冰冰推荐未来的合作对象。

```
1 starnet <- graph.edgelist(as.matrix(filmnet[, 1:2]), directed = F)
2 E(starnet)$weight <- filmnet[, 3]
3 V(starnet)$betweenness <- betweenness(starnet, directed = F)
4 actor <- data.frame(names(V(starnet)), unlist(degree(starnet)), V(starnet)
$betweenness)
5 names(actor) <- c("actor", "edges", "betweenness")
6 row.names(actor) <- NULL
7 temp <- neighbors(starnet, "范冰冰", mode = "total")
8 fanfriend <- unique(names(temp))
9 nonfriend <- actor[!(actor$actor %in% fanfriend), ]
10 nonfriend <- nonfriend[-which(nonfriend$actor == "范冰冰"),]
```

上述代码中，第 1 行代码构建新的合作网络；第 2 行代码赋值边线权重；第 3 行代码给节点创建新属性：中介度（betweenness），并使用 betweenness 中介度函数计算每一个节点的中介度，然后赋值；第 4 行代码提取节点名称、关系数和中介度，组成数据框 actor；第 5 行代码重命名数据框；第 6 行代码修改行名称为空值；第 7 行代码找出范冰冰所有的合作伙伴；neighbors 提取的仍然是个网络对象，所以第 8 行代码使用 names 函数提取合作伙伴的姓名；第 9 行代码找出非范冰冰的合作对象，即在范冰冰的合作伙伴中查找（%in%）actor，仅保留查不到的演员；第 10 行代码在非合作伙伴中剔除范冰冰自己。

- 使用均权法推荐好友

```

1 feifans <- nonfriend[, "actor"]
2 temp <- lapply(feifans, function(x, starnet, mode){
3   unique(names(neighbors(starnet, x, mode)))
4 }, starnet, "all")
5 cofriend <- lapply(temp, function(x, fanfriend){
6   fei <- unlist(x)
7   a <- length(x[x %in% fanfriend])
8 }, fanfriend)
9 cofriend <- unlist(cofriend)
10 nonfriend <- data.frame(nonfriend, cofriend)
11 temp <- nonfriend[, c("edges", "betweenness", "cofriend")]
12 temp <- apply(temp, 2, function(x) (x-min(x))/(max(x)-min(x)))
13 temp <- transform(temp, finalscore = edges + betweenness + cofriend)
14 finalscore <- temp$finalscore
15 totalscore <- data.frame(nonfriend, finalscore)
16 head(totalscore[order(totalscore$finalscore, decreasing = T),])

```

上述代码中，第1行代码提取非范冰冰合作伙伴，我们简称为非范好友；第2行代码使用 lapply 函数找出任一非范好友的好友，lapply 函数可以有很多个参数，第1个参数指定将要传递的数值，第2个参数用于指定可执行函数，用于接收传递过来的参数并执行，如果这个函数包括多个参数，这些参数可以依次作为 lapply 的第3个参数、第4个参数等传递，lapply 将 feifan 的每一个值（record）传递给匿名函数的第一个参数 x，匿名函数的另外两个参数（starnet、mode）通过 lapply 的第3个、第4个参数传递，在这种情况下，如果不传递必需的参数，lapply 内的执行过程不会在环境中寻找所需的参数，匿名函数内部取出每一个非范好友的合作伙伴，并去重，这样 temp 里面就存储了所有非范好友的好友，temp 是一个 list，list 的第一个元素是一个 vector，对应第一个非范好友的所有好友，依次类推，list 每一个元素与非范好友一一对应；第5行代码找出每一个非范好友与范冰冰共同好友的个数，即将 temp 的每一个元素作为参数 x 传递给匿名函数，匿名函数将 x 转化为 vector，然后与范冰冰的好友匹配（%in%），如果匹配到了，说明是共同好友，筛选出匹配到的 x 元素，并统计长度，就可以得到共同好友数了，lapply 的第3个参数 fanfriend（范冰冰好友）是为了作为匿名函数的第2个参数 fanfriend 传递；第9行代码将 list 解散为 vector 并重新赋值给 cofriend，这样就可以和非范好友一一对应了；第10行代码将非范好友和共同好友数捆绑为一个新的数据框 nonfriend。

所谓均权法就是认为每一个变量同等重要，不需要对特殊变量加权，但是这好像只是在理想状态下存在，很多时候，挖掘工作本质上就是为了加权。虽然我们不准备对关系数、中介度、共同好友数加权，但是它们的量纲不同将间接造成加权，所以我们首先使用归一法将三个变量的取值范围限制在 0~1 之间，第11行代码提取变量；第12行代码使用 apply 函数将 temp 的每一列传递给匿名函数，参数 2 表示将数据框的列作为参数传递，1 表示将数据框的行作为参数传递，匿名函数接受参数并执行归一化，这样 temp 存储了三列归一化后的数据；第13行代码使用 transform 函数在数据框 temp 中创建新列 finalscore，并将三列简单加和作为最终得分赋值给 finalscore；第14行代码将 temp\$finalscore 赋值给新向量 finalscore；第15行代码的 finalscore 与 nonfriend 捆绑在一起，形成新

## 探寻数据背后的逻辑：R 语言数据挖掘之道

的数据框；第 16 行代码通过 `order` 函数按照最终得分降序排序，查看结果的前 20 名，没错，我们给范冰冰推荐了一个“龙套王”——林雪先生。

到这里，我们基本上把社交网络的一些挖掘知识过了一遍，只是有没有收获就因人而异了，在数据挖掘中，算法不重要，构建问题和解释问题的过程才是重中之重，也就是我们所说的数据挖掘之道。

灵活使用统计知识解释并解决业务问题就是数据挖掘之道。

# 第 10 章

## 情感分析： 一种准确率高达 90%的新方法？

大多数数据挖掘方法和思路在很久之前就已经形成了，只是计算机的普及让这些创新的想法拥有了看似可行的技术支撑，给了它们起死回生的机会。情感分析的历史也可以追溯到 1950 年，没错，你真的没有看错，那时的情感分析用于研究论文评述的态度。如今，普及的社交媒体为情感分析提供了用武之地，文本数据源源不断地从微博、博客、微信、百度知道等交互平台中被抓取下来，而情感分析也被广泛地应用于挖掘互联网中的主观信息。

情感分析能够帮助企业掌握文本所表达的情感倾向：正向、负向或中性，帮助企业分析特定人群，特别是消费者，对某一类固定话题、产品、方法所持有的态度。企业可以使用挖掘出来的情感信息寻找新的营销机会，优化营销方式，针对目标人群采取精准营销等。

### 10.1 情感分析及其应用：这是老生常谈

在企业内，所有的数据挖掘工作归根结底都是解决业务问题，企业对这一点认识得越深刻，越能脚踏实地地解决问题，而不是制造虚假问题然后解决，或是答非所问，不能对问题实现有效的干预。

#### 10.1.1 情感分析的用途

情感分析有很多用途，如今网络社交越来越普及，“大 V”账号及意见领袖也越来越多，可以对商品和服务打分、评价的网站更是如雨后春笋，用户的评价和建议可以在全网传播，这些数据毫无疑问是精准营销的动力来源。企业可以根据情感分析发现新的市场机会，做好市场细分，进而推出成功的产品。但抓住这些评论有价值的部分是企业所面临的巨大挑战。情感分析的主要用途包括以

下 5 个方面。

### 1. 社交媒体监控

社交媒体监控可以非常有效地监控品牌的影响力，或识别对品牌有影响力的意见领袖，例如，如果要知道品牌负面评论的来源，则可以监控 50 个行业意见领袖的微博、博客，分析他们的微博或文章下面的用户评论，从而确定谁会引导企业的负向评论，之后可以和这些意见领袖合作，从而消除他们的负面评论带来的影响。

### 2. 改善或重塑公共关系

情感分析还能帮助企业改善或重塑公共关系策略，例如，可以探索销售热点和行业的流行趋势。同样，就像监控社交媒体一样，通过情感分析找到社会上对品牌有好感的意见领袖，与之建立并维持合作关系，这种方式显然是一种行之有效的公共关系管理策略。

### 3. 市场营销

情感分析正在逐步替代传统市场研究的内容，如用户情感需求调研、用户使用习惯及态度研究等。例如，三星想知道消费者对其新手机机型的评价，在社交媒体和数据挖掘兴起之前，它们只能通过市场调研的方式解决。而使用数据分析这种方式，则可以通过抓取消费者在各大消费网站（如亚马逊、京东、天猫及社交媒体）中留下的评论数据并加以分析，从而获知消费者对某个新手机机型的情感倾向，或者获得消费者对某个新手机机型的了解程度，以及消费者对手机机型的哪些属性比较赞赏，对哪些方面并不感兴趣。

### 4. 政策分析

分析微博、推特上的评论可以非常准确地了解大众对国家政策的理解和情感倾向。慕尼黑大学的研究表明，推特上的信息能够非常准确地反映选民的政治倾向，通过分析 2009 年德国总理大选期间涉及政党和政客的 10 万条推特，可以得到结论：推特的信息能够预示德国总理大选的结果，其准确性不亚于传统的民意调研。

### 5. 数据挖掘

情感分析还可以用于采集竞争对手的竞争优势，企业可以轻易地跟踪消费者在社交媒体中的情感倾向及竞争对手的情感倾向，了解消费者对竞争品牌的印象及对自己产品的情感倾向，这是非常好的竞争优势分析策略。更重要的是，情感分析的结果和情感指数还可以作为变量应用到其他数据挖掘项目中，例如，在预测用户流失率时，就可以关联用户的情感指数作为变量。

## 10.1.2 情感分析的方法论

受制于人们表达态度的方式异常复杂，情感分析面临着很多挑战，简单的方法难以识别真正的情感倾向。但常见的分析方法大致可以分为两个方向：基于词典的情感分析法和基于监督算法的情



感分析方法。

### 1. 基于词典的情感分析法

基于词典的情感分析法起源于基于语法规则的文本分析，其方法比较简单、纯朴。首先需要具有语法敏感性的专业人士构建情感分析的词典：正向情感词典和负向情感词典，即将某语言中用于表达情感的词汇分为两个类别，然后对比文本中正向情感词和负情感词的个数，评估文本的情感倾向，这种方法非常容易理解。

情感词也分轻重缓急，比如“喜欢”和“爱”虽然都是正向的，但程度是不一样的。因此，根据语言专家的分析，给予情感词不同的情感级别或权重，算得上是对上述分析方法的改进。毫无疑问，这种方法包含一定的语法分析成分，谷歌翻译的早期版本就是基于语法的，其效果可见一斑。

### 2. 基于监督算法的情感分析方法

基于监督算法的情感分析方法是首先制作一个规模庞大的训练数据集，由人工识别文本情感是正向的还是负向的，然后通过机器学习或算法（SVM、随机森林、朴素贝叶斯等）等方式训练模型，得出模型后再来识别新文本的情感倾向。此方法比较像垃圾邮件的分类方法：首先精挑一些垃圾邮件和正常邮件让模型学习，然后再将模型用于垃圾邮件的分选。

## 10.1.3 有关情感分析的一些知识和方向

首先我们应该感谢一些前辈所做的积累工作：在词典方面，中国知网的研究者整理出了知网情感 hownet 词典（更新地址：[http://www.keenage.com/html/c\\_bulletin\\_2007.htm](http://www.keenage.com/html/c_bulletin_2007.htm)），另外还有富士通公司的情感词典、程度词典等。

除了情感词典，情感分析语料库方面的前辈们也做了很多积累工作，比如清华大学李军标注的情感分析语料库等，以及 Tang 等人另辟蹊径，根据表情符号标注微博情感语料库类别。

基于多年从事文本挖掘和自然语言分析工作经验，在设计和评价情感分析方案时，我觉得应该考虑以下几点：

- （1）是否需要词典。
- （2）是否具有跨行业分析的功能。
- （3）使用什么算法，是分类任务，还是分值预测，这关系到情感分析的细腻度问题。
- （4）是否需要使用规则及其与程序速度的平衡关系。
- （5）是否关注情感归属问题，即主体词和情感的归属关系，是整体层次还是单个文本的个体层次。
- （6）准确率和速度。

以上是在设计和评价情感分析时需要考虑的几个基本问题，本章也尽量从这些方面来介绍。

## 10.2 文本分析的基本武器：R

工欲善其事，必先利其器，R 和 Python 都是数据挖掘的利器，虽然我们选择使用 R 做情感分析，但我认为 Python 更具后发优势，在网上搜索自然语言处理的相关内容，就会发现很多内容和 Python 有关。

### 10.2.1 RJava 包配置

既然选择了 R，就需要将 R 软件打造一番，首先需要安装 RJava 包，它是 R 和 Java 的通信接口，允许在 R 中直接调用 Java 的对象和方法，恰恰能满足很多包的需要，比如 Rwordseg。

在 Linux 环境下，只要 R 与 Java 的版本对应，即可使用常规方法安装 RJava 包。而在 Windows 环境下就比较麻烦，首先要安装 Java。可以在 Oracle 官网中下载 JDK 版安装包。注意，这里是 JDK 不是 JRE，至于选择 64 位版本还是 32 位版本，只要和 R 版本相同即可。

然后安装 Rtools(下载地址 <http://cran.csdb.cn/bin/windows/Rtools/>)，当然要选择和 R 匹配的 Rtools 版本。

完成以上工作后需要配置环境，在 Windows 7 中的“我的电脑”图标上单击鼠标右键，然后在弹出的快捷菜单中单击“高级系统设置”——“环境变量”命令，在环境变量中分别新建或添加相应的环境路径。

新建 classpath，并添加下面代码：

- classpath

```
1 .;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar;C:\Program Files\R\R-3.2.1\library\rJava\jri
```

其中最后一个值有时需要指定 JRI 的版本是 x64 还是 i386。

然后新建 JAVA\_HOME，并添加下面代码：

- JAVA\_HOME

```
1 C:\Program Files\Java\jdk1.8.0_51
```

再新建 Path，并添加以下代码。注意：所有的软件都要区分 i386 或 x64 版本。

- Path

```
1 C:\Rtools\bin;C:\Rtools\gcc-4.6.3\bin;%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin;C:\Program Files\R\R-3.2.1\bin\i386;C:\Program Files\R\R-3.2.1\bin\x64;C:\Program Files\Java\jdk1.8.0_51\jre\bin\server
```

最后新建 R\_HOME，并添加以下代码：

- R\_HOME

```
C:\Program Files\R\R-3.2.1
```

以上所有路径均是软件安装路径，其他不需要修改。

- 安装 rJava 包

```

1 if (!suppressWarnings(require("rJava"))) {
2   install.packages("rJava")
3   require("rJava")
4 }
5 .jinit()
6 s <- .jnew("java/lang/String", "Hello World!")
7 s
#[1] "Java-Object{Hello World!}"

```

如果 s 正常返回"Java-Object{Hello World!}"，则表示 RJava 包已经安装成功了。只有 RJava 包安装成功了，Rwordseg 包安装才可能成功，因为前者是后者的依赖包。

如果返回不正常，则说明 RJava 包还没安装好，需要重新安装。

## 10.2.2 Rwordseg 包安装

Rwordseg 是我接触最早的中文分词包，所以对其情有独钟。并不是因为它有多么优秀，而是我觉得分词是文本分析的基础工作，就其准确性而言，各种方法之间实在没有实质性的差别，因此一直在使用 Rwordseg 包。

在现实中，有一种现象很奇怪，比如，为了分词准确性提高 1%，有些人在评价结果时可能纠结于极个别的个体，这就本末倒置了。如果自变量是由个体组成的，则必然需要从群体上评价其准确性，而不是纠结于“黑天鹅事件”。对于防止“黑天鹅事件”，自然可以设计一套风险管理措施，比如可以分化赌注。下面继续安装 Rwordseg 包。

- 安装 Rwordseg

```

1 install.packages("Rwordseg", repos = "http://R-Forge.R-project.org")#这一步要完成需要安装 Rtools
2 library(rJava)
3 library(Rwordseg)
4 segmentCN("在此对原作者孙健表示强烈的敬意!")

```

如果在线安装不成功，则可以将安装包下载后在本地计算机中安装。

- 下载

```
1 https://r-forge.r-project.org/R/?group_id=1054
```

在本地计算机中安装 Rwordseg 很简单，如果使用的是 Rstudio 编辑器，则在其中单击“tools”选项，然后选择“install.packages”选项就可以安装。

## 10.2.3 jieba 分词包安装

虽然我比较执着于 Rwordseg 包，但并不代表读者也要执着于我的执着，这里推荐 jieba 分词包，其小巧玲珑，而且适用于 R 和 Python。除了词性标注等分词包必备功能外，jiebaR 中还加入了一些基础的文本分析算法，比如，提取关键字（TFIDF）、分析文本相似性等。

- 安装 jiebaR

```
1 library(devtools)
2 if (!suppressWarnings(require("Rcpp"))) {
3   install.packages("Rcpp")
4   require("Rcpp")
5 }
6 install_github("qinwf/jiebaR")
7 library(jiebaR)
8 initial <- worker()
9 seg <= "江州市长江大桥，参加了长江大桥的通车仪式。"
10 seg <= "H:/ 探寻数据背后的逻辑 R 语言数据挖掘之道 / 第 10 章情感分析
/rawdata/dataset_602123/ChnSentiCorp_htl_ba_2000/neg/neg.0.txt"
11 fenci <- segment("江州市长江大桥参加了长江大桥的通车仪式", initial)
```

如果使用 `install_github` 函数安装 Github 上的包，则需要调取 `devtools` 包，而该包需要调用 `Rtools` 包，这也是提前安装 `Rtools` 包的原因；另外，`jiebaR` 是使用 `Rcpp` 开发的，因此需要安装调用 `Rcpp` 包。

`jiebaR` 其实是一个刀架，里面的每一把刀都是一个分词引擎（有很多种），通过 `worker` 函数初始化引擎，建议不要更改默认设置（很多函数的默认设置必有其合理性），指定分词引擎后，只需要将文本通过“`<=`”符号赋值给分词引擎即可，其中有两种方式：一种是直接赋值一句话；另一种是指定文本路径，分词结果存放在相应的路径内；当然也可以像使用 `Rwordseg` 包的 `segmentCN` 函数一样使用 `segment` 函数分词，只不过后者需要指定分词引擎。

## 10.3 基于词典的情感分析的效果好过瞎猜吗

上面已经简单介绍了基于词典的情感分析的过程，即数据整理、词典整理、情感词匹配、计算情感得分和方法评估等。这种分析方法因其思路简单而遭人诟病，更有甚者认为其分析效果还赶不上瞎猜。真的是这样吗？

### 10.3.1 数据整理及词典构建

本节主要完成文本语料库和词典的整理，这里使用谭松波和清华大学李军等人标注的语料库进行分析，词典使用已经公开的词典资源，例如，中国知网等网站公布的情感词典。

#### 1. 数据整理

此语料库共包含两个数据集：清华大学李军标注的近 24000 个酒店评论文本和谭松波整理的 12000 个来自京东、携程、当当网的跨行业评论文本。李军的语料库（`review_sentiment`）包括两个数据集（训练数据集和测试数据集）和两个标注表（训练标注表和测试标注表），标注表注明了文本

的情感倾向，正向为 1，负向为-1。由于文本文件为分散的单个文本，所以需要批量读入文本文件。

- 获取文本路径

```
1 reviewpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdata/review_sentiment/train2"
2 completepath <- list.files(reviewpath, pattern = "*.txt$", full.names =
TRUE)
```

其中第 1 行代码设定了文件存放的路径。第 2 行代码使用 `list.files` 函数获取该路径下所有文件的文件名，如果 `full.names` 参数为真，则返回完整的路径，反之则返回文件名称；`pattern` 设置符合正则表达式的规则，仅提取符合要求的文件，防止读入系统文件，这里仅匹配以.txt 结尾的文件。

- 批量读入文本

```
1 read.txt <- function(x) {
2   des <- readLines(x)
3   return(paste(des, collapse = ""))
4 }
5 review <- lapply(completepath, read.txt)
#There were 50 or more warnings (use warnings() to see the first 50)
```

上面的代码先构造了一个 `read.txt` 函数，目的是将一个路径完整的 txt 文件加载进来，并将每个段落粘贴在一起，此函数是一个非常中规中矩的 R 函数模式：第 1 行代码为打开文件并分行读取（`readLines`），然后将每一行粘贴在一起，最后返回完整的文本内容。其中 `return` 声明返回的内容，如果不声明，则仅返回函数中形成的最后一个对象。`paste` 函数有两个指定间隔符号的参数：`sep` 和 `collapse`，前者将两个对象或两个对象的元素对应粘贴在一起时使用；后者是融合的意思，将向量或者 `list` 的元素粘贴在一起时使用。第 3 行代码使用了 `lapply` 函数，避免了循环读取每一个文本，保持了代码的整洁，提高了代码的运行速度。到这里，`review` 这个 `list` 就记录了文件下的所有文本内容，它的每一个元素都是一个文本的内容。如果你忍不住想用循环处理，那么说明你还处于初级阶段，不到万不得已绝对不能轻易使用 R 语言的循环语句。另外，如果程序发出警告，则可能是部分文本最后一行没有换行导致，文本的最后一行是换行符则标志一个文本正确结束，不用担心。

- list 转数据框

```
1 docname <- list.files(reviewpath, pattern = "*.txt$")
2 reviewdf <- as.data.frame(cbind(docname, unlist(review)),
                           stringsAsFactors = F)
3 colnames(reviewdf) <- c("id", "msg")
4 reviewdf$msg <- gsub(pattern = " ", replacement = "", reviewdf$msg)
5 reviewdf$msg <- gsub("\\t", "", reviewdf$msg) #有时需要使用\\t
6 reviewdf$msg <- gsub(",|", "", "", reviewdf$msg)
7 reviewdf$msg <- gsub("~|'", "", reviewdf$msg)
8 reviewdf$msg <- gsub("\\\\\"", "", reviewdf$msg)
# Warning message:
# In scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, :
# EOF within quoted string
```

在上面的代码中，第 1 行代码用于读取文件名称。第 2 行代码用 `unlist` 将 `review` 解散为 `vector`，并与文件名称按列（`cbind`）捆绑在一起成为一个新的数据框。第 3 行代码用于修改列名。第 4 行代码移除文本中的所有空格，第 1 个参数指定正则表达式模式，第 2 个参数指定替换为的内容。第 5、第 6 行代码替换所有的“\t”和半角逗号，因为 `csv` 格式的文档以半角逗号为分隔符半角，文中有半角逗号会报错，除半角逗号可能引起 `read.csv` 函数读取 `csv` 文件报错外，还有半角单引号（'）、半角双引号（"）、波浪号（~），都会引起读取文件时报错，导致 `csv` 文件或 `txt` 文件读取不完整。一旦提示“EOF within quoted string”，就要想办法找出这类符号并将其替换。因此，第 7 行代码依次替换了波浪号（~）和半角单引号（'），在它们之间用“|”符号隔开，表示或的关系。第 8 行代码替换所有的半角双引号（"），因为双引号在 R 语言中有特殊含义，所以要使用 3 个斜杠（\\）转义。上面处理了一下数据，下一步要给文本匹配上情感标注。

- 关联标注

```
1 reviewclass <- read.table("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/rawdata/review_sentiment/train2.rlabelclass")
                                stringsAsFactor = F)
2 colnames(reviewclass) <- c("id", "label")
3 library(plyr)
reviewdf <- join(reviewdf, reviewclass)
reviewdf <- reviewdf[!is.na(reviewdf$label),]
train <- reviewdf
```

在上面的代码中，`read.table` 读取训练集中的文本标注数据，该函数是 R 语言中读取数据的根函数，很多函数都是继承自 `read.table`，所以看到格式陌生的文件时，可以尝试使用这个函数读取。第 2 行代码更改了标注文档的列名称，`id` 列和 `reviewdf` 中的 `id` 列相同，`plyr` 包里的 `join` 函数会根据名称相同的列进行匹配关联，在 `join` 默认设置下执行左连接；然后将整理好的赋值给 `train` 备用。这样李军标注的训练数据集就整理完成了。

- review\_sentiment 测试数据整理

```
1 reviewpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/rawdata/review_sentiment/test2"
2 completepath <- list.files(reviewpath, pattern = "*.txt$", full.names = TRUE)
3 review <- lapply(completepath, read.txt)
4 docname <- list.files(reviewpath, pattern = "*.txt$")
5 reviewdf <- as.data.frame(cbind(docname, unlist(review)),
                                stringsAsFactors = F)
6 colnames(reviewdf) <- c("id", "msg")
7 reviewdf$msg <- gsub(pattern = " ", replacement = "", reviewdf$msg)
8 reviewdf$msg <- gsub("\\t", "", reviewdf$msg) #有时需要使用\\t
9 reviewdf$msg <- gsub(",|,", "", reviewdf$msg)
10 reviewdf$msg <- gsub("~|'", "", reviewdf$msg)
11 reviewdf$msg <- gsub("\\\\\"", "", reviewdf$msg)
12 reviewclass <- read.table("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情
```

```
感分析/rawdata/review_sentiment/test2.rlabelclass", stringsAsFactor = F)
13 colnames(reviewclass) <- c("id", "label")
14 library(plyr)
15 reviewdf <- join(reviewdf, reviewclass)
16 reviewdf <- reviewdf[!is.na(reviewdf$label),]
17 test <- reviewdf
```

整理测试数据集使用的代码与整理训练数据集相同。

dataset\_602124 数据集分为“当当”“京东”“携程”3 个子文件夹，它们的下面又分别有“pos”和“neg”文件夹。顾名思义，“pos”文件夹中全是情感正向的文本；“neg”文件夹中则为负向的文本，为了操作简便，分别把文本统一整理到 pos 和 neg 文件夹内，然后在进行数据处理。

- 获取 pos 文本路径

```
1 reviewpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdata/dataset_602124/pos"
2 completepath <- list.files(reviewpath, pattern = "*.txt$", full.names =
TRUE)
```

上面的代码用于读取 dataset\_602124 数据集，设定文件路径。

- 批量读入 pos 文本

```
1 review <- lapply(completepath, read.txt) #自编函数 read.txt
2 docname <- list.files(reviewpath, pattern = "*.txt$")
3 docname <- paste("p", docname, sep = "")
4 label <- rep(1, length(docname))
5 reviewdf <- as.data.frame(cbind(docname, unlist(review), label),
stringsAsFactors = F)
6 colnames(reviewdf) <- c("id", "msg", "label")
7 reviewdf$msg <- gsub(pattern = " ", replacement = "", reviewdf$msg)
8 reviewdf$msg <- gsub("\t", "", reviewdf$msg) #有时需要使用\\t
9 reviewdf$msg <- gsub(",|,", "", reviewdf$msg)
10 reviewdf$msg <- gsub("~|'", "", reviewdf$msg)
reviewdf$msg <- gsub("\\\\\"", "", reviewdf$msg)
pos6 <- reviewdf
```

在上面的代码中，第 1~3 行代码不多说了。第 4 行代码为读入的数据添加情感倾向标签，正向数据全部标注为 1，负向数据全部标注为-1（暂时注释掉了），rep 函数复制出一个和向量 docname 等长的向量。第 5、第 6 行代码按列将 docname、unlist(review)、label 捆绑生成数据框，并重命名，其顺序、名称和 train 数据框保持一致。第 7~10 行代码就不多说了。最后 reviewdf 赋值给 pos6，pos 文件就算整理完了。

- 整理 neg 文本

```
1 reviewpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdata/dataset_602124/neg"
2 completepath <- list.files(reviewpath, pattern = "*.txt$", full.names =
TRUE)
3 review <- lapply(completepath, read.txt) #自编函数 read.txt
```

```
4 docname <- list.files(reviewpath, pattern = "*.txt$")
5 docname <- paste("n", docname, sep = "")
6 label <- rep(-1, length(docname))
7 reviewdf <- as.data.frame(cbind(docname, unlist(review), label),
                           stringsAsFactors = F)
8 colnames(reviewdf) <- c("id", "msg", "label")
9 reviewdf$msg <- gsub(pattern = " ", replacement = "", reviewdf$msg)
10 reviewdf$msg <- gsub("\t", "", reviewdf$msg) #有时需要使用\\t
11 reviewdf$msg <- gsub(",|", "", reviewdf$msg)
12 reviewdf$msg <- gsub("~|'", "", reviewdf$msg)
13 reviewdf$msg <- gsub("\\\\\"", "", reviewdf$msg)
14 neg6 <- reviewdf
15 temp <- rbind(pos6, neg6)
```

整理 neg 文本的代码与整理 pos 文本的代码基本相同,其中最后一行代码将 pos 和 neg 文本中的数据按行粘贴在一起。

- 输出训练数据集和测试数据集

```
1 train <- rbind(train, temp)
2 write.csv('H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/train.csv', row.names = FALSE)
3 write.csv('H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/test.csv', row.names = FALSE)
```

在上面的代码中,第 1 行代码将 train 和 temp 按行粘贴(rbind)在一起,生成用于训练的语料库 train,测试的语料库仅仅使用李军的测试数据集 test 即可。只需要将 train 和 test 输出到专用的数据存储文件夹 data,将 write.csv 输出到语料文档,输出格式为.csv,参数 row.names 设置为非,表示不输出行编号。

这样测试数据集和训练数据集语料库都已经准备好了,并存储在 data 文件夹里备用。

## 2. 词典构建

尽管我们搜集了很多部情感词典,经过挑选,这里暂时整合中国知网、台湾大学、清华大学和一部未标注来源的词典,再次感谢这些词典的作者。首先我们需要将这些词典整合为正向情感词集 pos 和负向情感词集 neg。下面已经将各个词典的正向汇和负向词汇整理到两个文件夹内。

- 正向词汇整理

```
1 dictpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdict/posdic"
2 completepath <- list.files(dictpath, pattern = "*.txt$", full.names = TRUE)
3 dict <- lapply(completepath, readLines)
4 dict <- unique(unlist(dict))
5 pos <- dict
6 write.csv(pos, 'H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/pos.csv', row.names = FALSE)
```



在上面的代码中,第 1 行和第 2 行代码不多说了,第 3 行代码在 `lapply` 函数中直接使用了 `readLines` 函数,将每个文件按行读取。第 4 行代码将 `list` 解散 (`unlist`) 成为一个向量,各个词典中肯定有相互重复的词,所以使用 `unique` 函数去重,这样词典的正向词汇就整理完成了。

- 负向词汇整理

```
1 dictpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdict/negdic"
2 completepath <- list.files(dictpath, pattern = "*.txt$", full.names = TRUE)
3 dict <- lapply(completepath, readLines)
4 dict <- unique(unlist(dict))
5 neg <- dict
6 write.csv(neg, 'H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/neg.csv', row.names = FALSE)
```

负向词汇的整理方法与正向词汇相同,最后将正向词汇和负向词汇输出到 `dict` 文件夹中即可。

### 10.3.2 分词整理

实际上,基于词典的情感分析的算法和模型先入为主地规定了:统计文本正、负情感词的得分之和,如果得分为正,则文本情感倾向为正,反之亦然。所以,不需要训练模型,直接使用测试数据集测试一下即可。

基于以上规则,首先要进行中文分词,在分词之前要将文本预处理,包括清除一些英文和数字等。

- 分词预处理

```
1 test <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/rawdata/review_sentiment/test.csv", sep = ",", header = T, stringsAsFactors =
F)
2 sentence <- as.vector(test$msg)
3 sentence <- gsub("[[:digit:]]*", "", sentence) #清除数字[a-zA-Z]
4 sentence <- gsub("[a-zA-Z]", "", sentence)
5 sentence <- gsub("\\\\.", "", sentence)
6 test <- test[!is.na(sentence), ]
7 sentence <- sentence[!is.na(sentence)]
8 test <- test[!nchar(sentence) < 2, ]
9 sentence <- sentence[!nchar(sentence) < 2]
```

在上面的代码中,第 1 行代码用于读取 `csv` 文件,并设置 `stringsAsFactors` 参数为非,即不要把字符转化为因子。第 2 行代码将文本内容转化为向量 `sentence`。第 3 行代码清除数字。第 4 行代码清除英文字符。由于某些文档可能是由外国人写的,全部是英文,经过以上几步处理就只剩下了 `dot` 符号,所以第 5 行代码将这类符号清除。经过以上处理,可能一些文本已经变成了空值或者小于两个字符了,所以第 6 行代码将原数据框中这些空值文本清除。第 7 行代码将对应的 `sentence` 里的空值清除。

注意：第 6 行代码根据 sentence 是否为空值完成筛选的，所以要先剔除数据框 test 内的空值，然后再剔除 sentence 内的空值。最后两行代码筛出字符数小于 2 的文本，nchar 函数用于对字符计数，英文叹号“!”为 R 语言里的“非”函数。

另外，既然整合了大量的词典，就要尽量保证分词器能够把这些情感词汇分出来，所以需要情感词典添加到分词器的词典中，虽然这种方法在特殊情况下并不一定奏效，但至少增加了分词器在分词时的计算权重。

- 添加词典

```
1 pos <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/pos.csv", header = T,
  sep = ",", stringsAsFactors = F)
2 weight <- rep(1, length(pos[,1]))
3 pos <- cbind(pos, weight)
4 neg <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/neg.csv", header = T,
  sep = ",", stringsAsFactors = F)
5 weight <- rep(-1, length(neg[,1]))
6 neg <- cbind(neg, weight)
7 posneg <- rbind(pos, neg)
8 names(posneg) <- c("term", "weight")
9 posneg <- posneg[!duplicated(posneg$term), ]
10 dict <- posneg[, "term"]
11 library(Rwordseg)
12 insertWords(dict)
```

在上面的代码中，第 1 行代码读取正向情感词。第 2 行代码创建权重向量，这里任何一个正向词汇的权重均为 1，负向词汇的权重为-1。第 3 行代码为 pos 添加权重列。第 4~6 行代码的作用同上，为负向情感词添加权重列。第 7 行代码将正负情感词按行粘贴在一起。第 8 行代码更改列名称。因为各个词典对情感词的倾向定义可能矛盾，所以会出现同一个词具有正向情感和负向情感两种倾向的情况，尽管这种情况更加符合现实，但是违背了基于词典的情感分析的原假设，所以要将这些词去重。这里使用的方法是，如果一个词同时属于正向情感词和负向情感词，那么仅保留正向分类。第 9 行代码中 duplicated 函数的作用和 unique 函数比较相似，它返回重复项的位置编号，比如“爱”这个词在数据框和向量中第二次及更多次出现的位置，第一次出现的位置不返回，如果加了“非”函数，则表示仅保留第一次出现的词汇。第 10~12 行代码提取 posneg 的 term 列并使用 Rwordseg 包添加自定义词典的函数 insertWords，将词典添加入分词器。

下面就可以分词了。使用 Rwordseg 包中的分词函数 segmentCN，仅需要将向量、字符串或者文本文件地址赋值给参数 strwords 即可。结果是一个和原向量等长的 list。List 中的每一个元素是一个词语字符向量，对应原来的文本向量，对于函数的其他参数仅需要注意 nosymbol，指定是否保留文本中的符号。

- 分词

```
1 system.time(x <- segmentCN(strwords = sentence))
2 temp <- lapply(x, length)
3 temp <- unlist(temp)
4 id <- rep(test[, "id"], temp)
5 label <- rep(test[, "label"], temp)
6 term <- unlist(x)
7 testterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)
```

在上面的代码中，第 1 行代码使用分词函数 `segmentCN` 分词，其结果是一个和 `sentence` 等长的 `list`，`list` 中的每一个元素对应文本的分词结果。这里使用了 `system.time` 函数返回代码块的执行时间，建议每次可能耗费时间较长的过程，都要使用少量数据预估一下时间，这是一个非常好的习惯。但是需要将分词结果和文本的 `id` 关联上，那么文本分出多少个词就要重复多少次文本 `id`，所以第 2 行代码使用 `lapply` 函数返回 `x` 中每一个元素的长度，即文本分出多少个词。`lapply` 函数返回的是一个 `list`，所以第 3 行代码使用 `unlist`。第 4 行代码将每一个对应的 `id` 复制相应的次数，就可以和词汇对应了。第 5 行代码将 `id` 对应的情感倾向标签复制相同的次数。第 6 行代码将 `list` 解散为向量。第 7 行代码将一一对应的 3 个向量按列捆绑为数据框，分词整理就结束了。

在分析过程中，难免会产生很多中间变量，它们会占用大量内存，虽然 `dplyr` 包的管道函数看似解决了这个问题，但是毕竟用处不广。我通常会把所有的临时中间变量命名为 `temp`，只需要保证在下一个 `temp` 出现之前，临时变量不会再被沿用就可以了，如上面的 `temp`，这样中间变量始终只有一个，即 `temp`。

虽然算法已经足够简单，没有必要去除停用词，但是为了谨慎，文本分析中的每一个环节都不能少，这里还是认真地去除停用词。

- 去除停用词

```
1 stopword <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/stopword.csv", header = T, sep = ",", stringsAsFactors = F)
2 stopword <- stopword[!stopword$term %in% posneg$term,]
3 testterm <- testterm[!testterm$term %in% stopword,]
```

在上面的代码中，第 1 行代码为读取停用词典。在停用词中有可能有一些词有感情色彩，既然要去除停用词，首先要将这类具有情感色彩的停用词从停用词典中去除，第 2 行代码使用函数 `%in%` 在 `posneg$term` 中查找 `stopword` 的元素，如果查到了就返回真值，如果没查到就返回假，结果是一个和 `stopword` 等长的布尔值向量，“非”函数将布尔值反向。第 3 行代码为去除停用词。

### 10.3.3 情感指数计算

现在有了分词表 `testterm` 和情感词典 `posneg`，另外，前面已经给情感词添加了权重列，所以，只需要匹配 `term` 列，即可给 `testterm` 表关联上情感权重，然后就可以计算文本的情感得分了。需要注意的是，如果一个情感词出现多次，则其权重加倍，当然也可以尝试对词汇去重，即每个情感词

只记录一次，也许会有出乎意料的结果。

- 关联情感词权重

```
1 library(plyr)
2 testterm <- join(testterm, posneg)
3 testterm <- testterm[!is.na(testterm$weight), ]
4 head(testterm)
```

在上面的代码中，使用 plyr 包的 join 函数进行左关联，给 testterm 表关联上情感权重，然后筛选那些没有权重的非情感词汇。让人奇怪的是，很多看似不具有情感色彩的词，这些词典竟然也把它们当成了情感词，看来还需要优化情感词典。

- 计算情感指数

```
1 dictresult <- aggregate(weight ~ id, data = testterm, sum)
2 dictlabel <- rep(-1, length(dictresult[, 1]))
3 dictlabel[dictresult$weight > 0] <- 1
4 dictresult <- as.data.frame(cbind(dictresult, dictlabel), stringsAsFactors = F)
```

在上面的代码中，第 1 行代码使用透视表函数 aggregate 对 weight 列以文本 id 分组求和，得出了每个文本的情感得分，其中情感得分大于 0，倾向为正，标记为 1，反之则标记为-1。第 2 行代码使用 rep 函数产生一个和 dictresult 等长的-1 向量。第 3 行代码为筛选情感得分大于 0 的文本，然后根据返回的布尔向量进行筛选，将向量 dictlabel 相应的位置更改为 1。第 4 行代码将向量 dictlabel 和数据框 dictresult 捆绑在一起，dictlabel 列即基于词典的情感分析法给出的文本标签。

### 10.3.4 方法评价：优、缺点分析

基于词典的情感分析方法基本上省略了建模过程，也就用不着交叉检验了，只需要统计人工标记和词典标记的交叉表就可以反映出结果的准确性了。

- 交叉表评价

```
1 temp <- unique(testterm[, c("id", "label")])
2 dictresult <- join(dictresult, temp)
3 evaluate <- table(dictresult$dictlabel, dictresult$label)
#      -1      1
# -1  399    35
#  1 1558 1904
```

在上面的代码中，第 1 行代码用于筛选测试数据的文本 id 和人工标签 label，并去重。第 2 行代码将 temp 和分类结果通过文本 id 左关联。第 3 行代码使用 table 函数对人工标注列和词典标注列做交叉表，返回结果的准确率仅仅达到 59.1%。但从执行的过程中我们也发现，很多不具有情感色彩的词汇被定义为了情感词，例如“的”“了”“还”“在”“我”、“都”、“把”、“上”等，这些词汇都是高频词汇，而上面的计算方法按照词汇出现频次重复计算，所以导致结果与实际偏差很大。

暂时的改进办法是修改优化词典，去除这类词汇，或者更改为去重计算，即一条评论中某个词

汇无论出现多少次都只计算一次权重。下面将源代码在优化后的词典中执行一次看看结果，代码无须更改，只需要将正负情感词典地址改为“自备词典”文件夹里的词典即可。

- 词典稍优化的结果

```
1 evaluate
#      -1      1
#   -1  993  287
#    1  965 1648
```

从结果中可以看出，简单地优化了一下词典，结果的准确率就上升到了 67.8%，所以从各个方面考量，基于词典的情感分析方法仍然有提升的空间，比如细化词汇的权重等。

毫无疑问，如果不追求“高大上”的算法，那么基于词典的分析方法不失为一种好的情感分析方法。其实，有时候我们使用了很多方法，最后发现结果并没有什么发生质变，也浪费了大量时间。比如，在优化词典的时候，我希望使用“高大上”的算法解决问题，自动分辨出情感词，我尝试了卡方统计量、各种分类器等，结果可想而知，浪费了大量的时间，最后还是使用人工的方法将词典优化了一遍。是的，用肉眼分析！其实，有时候看起来最笨的方法也许是现阶段最有效、最合适、最省事的方法，只是它看起来很低级，这也许就是笨方法的高深之处，“聪明人”是不屑于使用这些方法的。

另外，仅仅使用词汇并不能非常准确地识别一条文本所表达的情感倾向。一些修辞手法，例如反讽、欲扬先抑等，也会让基于词典的情感分析变得困难。

## 10.4 监督式情感分析：挑选训练数据集是所有人心中的痛

除了基于词典进行情感分析，现在比较流行用监督式分析模型进行情感分析，在这个领域中可以使用各种“高大上”的算法，如 SVM、决策树、随机森林、人工神经网络等。但效果怎么样还是有待检验。

一个完整的监督式分析模型一般分为以下步骤：（1）构建训练库，监督式过程必须通过训练数据集帮助模型识别特征。（2）提取特征及构建模型，有时候在构建模型之前，为了减少模型的复杂度，可能使用其他方法识别一些特征或特征词，比如 TFIDF 算法，但也许会出现帮倒忙的现象，反而使模型准确度大幅下降，尽管减少了建模的复杂度，但增加了整个分析过程的复杂度。（3）使用  $k$  层交叉检验进行结果评价。

### 10.4.1 TFIDF 指标

既然决定使用监督式分析模型，就要有一套衡量特征变化的指标，比如可以使用词频（Term Frequency, TF）作为变量，也可使用文档频率（Document Frequency, DF）作为变量。总之，在构建模型之前需要构建几个指标（这是文本分析的基本特征），我们不妨暂定 TF、DF 和 IDF 作为备选

指标。

下面介绍一下 TFIDF，方便读者理解。假设有一篇《中国抗战阅兵时间表》的文章，现在要提取这篇文章中的关键词。比较简单、有效的方法就是根据词频 TF 提取。TF 就是一篇文章中出现某个词的次数（当然如果是进行文章之间的横向比较，因为文章有长有短，则还要除以文章的总词数）。你可能认为“中国”出现的次数最多，其实不然，“的”“是”“在”“地”之类词出现的次数是最多的，这类词是停用词，在提取关键词之前必须剔除。

- 1 TF = 某词在文章中出现的次数
- 2 TF = 某词在文章中出现的次数 / 文章包含的总词数

剔除这些词之后“中国”就成了出现最多的词，但是看一下题目会发现，“中国”在所有的文章中出现的可能性都很大，而“抗战”“阅兵”“时间表”才是更关键的关键词，所以要给 TF 指标一个权重。于是“逆文档频率”（Inverse Document Frequency， IDF）横空出世，但首先需要知道文档频率（Document Frequency）是什么，二者计算公式如下：

- 1  $DF = (\text{包含某词的文档数}) / (\text{语料库的文档总数})$
- 2  $IDF = \log((\text{语料库的文档总数}) / (\text{包含某词的文档数} + 1))$

如果一个词比较“常见”(指在日常所有文档中),那么它的 IDF 指标就比较低。要计算 IDF 指标,首先要有一个充实的语料库。利用 IDF 作为惩罚权重,就可以计算词的 TFIDF:

$$1 \text{ TFIDF} = \text{TF} * \text{IDF}$$

这样比较常见的“中国”在《中国抗战阅兵时间表》这篇文章中的 TFIDF 就会被拉低, 而其他稀有词汇的 TFIDF 就会相对提高, 这样在按照 TFIDF 排序时即可以方便地找出关键词了。

通过计算 TF、DF、TFIDF 这 3 个指标构建语料库, 至少保证了监督模型的输入变量, 可以选择其中一种或多种作为指标构建模型。

### 10.4.2 构建语料库

在建模之前，应该使用训练数据集构建一个语料库，语料库中至少包括以下几个维度：文本 id、词汇、TF、DF 和 TFIDF。首先要将训练数据集清洗一下，清洗过程基本和基于词典的情感分析方法相同，但是这里好像碰到了一个比较棘手的问题。

- 训练数据集分词预处理

```
1 train <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/train.csv", sep = ",", header = T, stringsAsFactors = F)
# Warning message:
# In scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, :
#   EOF within quoted string
2 sentence <- as.vector(train$msg)
3 sentence <- gsub("[[:digit:]]*", "", sentence) #清除数字[a-zA-Z]
4 sentence <- gsub("[a-zA-Z]", "", sentence)
5 sentence <- gsub("\\.\\.\\.\\.", "", sentence)
6 train <- train[!is.na(sentence), ]
```

```

7 sentence <- sentence[!is.na(sentence)]
8 train <- train[!nchar(sentence) < 2, ]
9 sentence <- sentence[!nchar(sentence) < 2]

```

有时候在使用 read.csv 函数读取文件时，可能会报警：“EOF within quoted string”，一般此情况为数据中有不正常的符号所致，常见的方法是将 quote 设置为空值。这样做虽然避免了报警，但是仍然解决不了问题，有时数据会对不上号，所以，最好将一些特殊符号去除。前面已经讲过这类符号，但可能并没有完全总结。

- 训练数据集分词

```

1 library(Rwordseg)
2 insertWords(dict)
3 system.time(x <- segmentCN(strwords = sentence))
4 temp <- lapply(x, length)
5 temp <- unlist(temp)
6 id <- rep(train[, "id"], temp)
7 label <- rep(train[, "label"], temp)
8 term <- unlist(x)
9 trainterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)

```

分词的整理过程和 10.4.1 节一样，另外，这里沿用了 10.4.1 节优化的情感词典 dict。其实在这一步和上一步都可以编写相应的自编函数处理，但是建议不要轻易选择一个规模庞大的函数，原因有两个，其一，代码不易读；其二，如果函数处理过程的一般性不强，那么或许会适得其反，每次用时都要修修补补地自编函数还不如分步处理好。

- 去除停用词

```

stopword <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/stopword.csv", header = T, sep = ",", stringsAsFactors = F)
stopword <- stopword[!stopword$term %in% dict,]
trainterm <- trainterm[!trainterm$term %in% stopword,]

```

在使用复杂的算法时，尽量去除一些非特征词汇可以有效地降低计算量和内存占用率。尽管在基于词典的情感分析方法中去除停用词这个环节可有可无，但是这里将要用到其他复杂的算法，去除停用词是必需环节，因此，这里使用 10.4.1 节优化的情感词典去除停用词中具有情感色彩的成分，然后对分词结果去除停用词。

- 计算 TF、DF 指标

```

1 trainterm <- trainterm[grepl("\\S", trainterm$term),]
# trainterm[!grepl("\\S", trainterm$term),] #通过这句可以查看这种空白符
# 例如
# segmentCN("")
2 trainterm$logic <- rep(1, nrow(trainterm)) #添加辅助列
3 library(dplyr)
4 traintfidf <- aggregate(logic ~ id + label + term, data = trainterm, FUN
= sum) %>% rename(tf = logic)
5 total <- length(unique(traintfidf$id))

```

```
6 temp <- data.frame(table(traintfidf$term)/total)
7 names(temp) <- c("term", "df")
8 traintfidf <- left_join(traintfidf, temp)
```

在统计 TFIDF 等指标之前，还要处理一下数据。因为在分词的时候分出了空白符，这种空白符既不能用 `is.na`、`is.null`、`is.nan` 这些函数查出来，也不能使用常见的空白符（空格" "，制表符"`\t`"，换行符"`\n`"，回车符"`\r`"，垂直制表符"`\v`"，分页符"`\f`"，包括空白符（"`\\s`"）等正则规则查出来，所以在上述代码中，第 1 行代码使用非空白符（"`\\S`"）筛除这种情况。除了在分词时会碰到这种情况，在抓取网络数据时也会遇到。

要统计 TF 指标，可以通过 `table` 函数、`dcast` 函数（`reshape2` 包、`plyr` 包中都有这个函数）等实现，但是尝试之后发现它们不是速度慢，就是占用内存太高，包括 `data.table` 包里的 `dcast` 函数，原因在于它们的中间过程要进行矩阵的转换。这里使用 `aggregate` 统计每篇文章每个词的频次。第 2 行代码添加了一个辅助列 `logic`，当然不添加辅助列，设置 `aggregate` 里的 `FUN` 参数为 `length` 也能完成，但是数据量大时耗费时间太长，不如添加辅助列，而 `FUN` 参数调用 `sum` 函数速度快，这句代码的意思就是按照 `id`、`term`、`label` 三列分组后对 `logic` 求和。

第 5 行代码统计出参与计算的文章 `id` 数，即总文章数。第 6 行代码使用 `table` 函数计算文档频数，然后除以总文档数得出文档频率（`df`）。第 7 行代码进行重命名。第 8 行代码使用 `dplyr` 包里的 `left_join` 函数将 `traintfidf` 与 `temp` 左关联。这里需要提醒读者，不要将 `dplyr` 包、`plyr` 包同时使用，比如在这里就会导致 `rename` 函数被覆盖，二者的功能相似，没必要同时加载，可以先加载 `plyr` 再加载 `dplyr`。

- 计算逆文档频率（IDF）

```
1 temp <- data.frame(log(total/(table(traintfidf$term) + 1)))
2 names(temp) <- c("term", "idf")
3 traintfidf <- left_join(traintfidf, temp)
4 traintfidf$idf <- traintfidf$tf*traintfidf$idf
```

在上面代码中，第 1 行代码按照逆文档频率的计算公式计算 IDF 指标。第 2 行代码重命名 `temp` 数据框。第 3 行代码进行左关联。第 4 行代码计算 TFIDF 指标。到这里基本上有用的指标都用到了。

### 10.4.3 随机森林模型

现在有了 3 个指标：TF、DF 和 TFIDF，选用哪个指标用于构建模型呢？由于 TF 受高频词影响较大，这里暂时将其排除。根据上面的统计逻辑发现，某个词在正向样本中的 DF 指标和负向样本相同，因为我们并没有把正向样本和负向样本分开统计，所以在这种情况下使用 DF 指标建模基本上不可能将正向样本和负向样本分开，只有选用 TFIDF 指标了。构建模型时需要将每一个词汇作为一个变量或者维度，这样矩阵会变得异常稀疏，但是这里先不用在意这些，在企业内做数据挖掘建模时，第一目标不是追求模型统计的完美性，而是追求测试数据集和训练数据集的稳定性和准确性。

#### 1. 构建随机森林模型及计算情感分析指数

随机森林模型既能完成分类任务也能完成回归预测任务，训练数据标签里只有两个分类；1（正



向) 或 -1 (负向), 在理论上属于分类任务, 但是情感分析在未来肯定要追求更加细腻的分类, 到达一定程度后最终将由分类任务演化为回归预测任务, 而这里暂时使用随机森林完成分类任务, 读者也可以尝试完成回归预测任务。

在构建随机森林模型之前, 首先要把数据调整为 randomForest 包要求的格式: 即为数据框或者矩阵, 需要将原来的数据框调整为以每个词作为列名称 (变量) 的数据框。

- 模型构建

```
1 library(reshape2)
2 train <- dcast(data = traintfidf, id + label ~ term, sum, value.var = "tfidf")
3 # Error: std::bad_alloc
4 library(randomForest)
5 row.names(train) <- train[, "id"]
train <- subset(train, select = -id)
train$label <- as.factor(train$label)
6 Randommodel100 <- randomForest(x = subset(train, select = -label), y =
train[, "label"], importance = TRUE, proximity = FALSE, ntree = 100)#构建模型
print(Randommodel100)
# Call:
# randomForest(x = subset(train, select = -label), y = train[, "label"],
ntree = 100, importance = TRUE, proximity = FALSE)
#
# Type of random forest: classification
#
# Number of trees: 100
# No. of variables tried at each split: 157
#
# OOB estimate of error rate: 7.04%
# Confusion matrix:
#      -1      1 class.error
# -1 11602  274  0.02307174
# 1   968 4808  0.16759003
```

在上面的代码中, 第 2 行代码将原来的 long 型数据框转化为 wide 型数据框, data.table 包里的 dcast 函数比 reshape2 包里的 dcast 函数好用, 尽管它们的参数都一样, 但是很多人还是比较喜欢 reshape2 包。然而这一步需要耗费大量的计算机内存, 本书的代码是在服务器上运行完成的。如果你的计算机报告内存不足, 则可以使用 data.table 包里的 dcast 函数试试。第 4~5 行代码用于移除 id 列, 因为 id 列不是随机森林模型所需要的数据, 所以先将 id 列赋值给行编号, 然后将其移除, 这样既对原文本做了标记, 同时移除了对于建模无用的数据。第 6 行代码将标签列转化为因子, randomForest 函数在执行建模任务时, 首先判断因变量的类型, 如果因变量是因子, 则执行分类任务, 如果因变量是连续型变量, 则执行回归预测任务。第 7 行代码执行建模, x 参数设定自变量数据集, y 参数设定因变量数据列, importance 设定是否输出因变量在模型中的重要性, 如果移除某个变量, 则模型方差增加的比例是它判断变量重要性的标准之一, proximity 用于设定是否计算模型的临近矩阵, ntree 用于设定随机森林的树数, 最后代码输出模型在训练集上的效果。

从上面的结果中可以看到，结果的准确率达到 90% 以上，使用随机森林模型对训练集的分类还是不错的。另外，有人说随机森林不可能过拟合。下面验证一下上面的随机森林模型是否发生了过拟合。所谓过拟合，就是模型在训练数据集上预测比较准确，在测试数据集上预测效果就不准确的现象。

## 2. 测试随机森林模型

一万次过分小心都不为过，莽撞送死一次也就够了。

——《射雕英雄传》

使用 1 万条数据对模型测试 10 万次都不为过，首先使用测试数据集测试一下随机森林模型的准确率。建议在模型监控和评价的过程中，使用基础的统计指标，这样通俗易懂，而且计算量比较小。测试数据集与训练数据集一样，要经过分词、去除停用词、计算指标等过程。

- 测试数据集分词整理

```
1 test <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/test.csv", sep = ",", header = T, stringsAsFactors = F)
2 sentence <- as.vector(test$msg)
3 sentence <- gsub("[:digit:]*", "", sentence) #清除数字[a-zA-Z]
4 sentence <- gsub("[a-zA-Z]", "", sentence)
5 sentence <- gsub("\\.\\.\\.\"", "", sentence)
6 test <- test[!is.na(sentence), ]
7 sentence <- sentence[!is.na(sentence)]
8 test <- test[!nchar(sentence) < 2, ]
9 sentence <- sentence[!nchar(sentence) < 2]
10 s_ymtime(x <- segmentCN(strwords = sentence))
11 temp <- lapply(x, length)
12 temp <- unlist(temp)
13 id <- rep(test[, "id"], temp)
14 label <- rep(test[, "label"], temp)
15 term <- unlist(x)
16 testterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)
17 testterm <- testterm[!testterm$term %in% stopwords,]
```

上面的代码几乎和训练数据集中的代码一样，这里不再分开讲了，下面需要计算测试数据集的 3 个重要指标，其中 DF 和 IDF 这两个指标并不是基于测试数据集计算的，从这两个指标的概念上分析，它们的值基于语料库定义而不是某个小集合，只是我们的语料库刚好既是训练数据集又是语料库而已（因为数据太少）。试想一下，如果你的测试数据集每次只有一条数据，那它的 DF 和 IDF 指标情感是不是永远不变？

- 测试数据集指标计算

```
1 testterm <- testterm[grep("\\S", testterm$term),]
2 testterm$logic <- rep(1, nrow(testterm))# 添加辅助列
3 testtfidf <- aggregate(logic ~ id + label + term, data = testterm, FUN =
```

```

sum) %>% rename(tf = logic)
# df 来源于语料库
4 total <- length(unique(traintfidf$id))
5 temp <- data.frame(table(traintfidf$term)/total)
6 names(temp) <- c("term", "df")
7 testtfidf <- left_join(testtfidf, temp)
# idf 来源于语料库
8 temp <- data.frame(log(total/(table(traintfidf$term) + 1)))
9 names(temp) <- c("term", "idf")
10 testtfidf <- left_join(testtfidf, temp)
11 testtfidf$tfidf <- testtfidf$tf*testtfidf$idf

```

在上面的代码中，前 3 行代码用于计算 TF 指标（TF 指标是针对单个文档的）。第 4~6 代码行用于计算语料库中词语的 DF 指标。第 7 行代码给测试数据集匹配上 DF 指标。第 8~9 行代码计算语料库中词语的 IDF 指标。第 10 行代码给测试数据集匹配上 IDF 指标。第 13 行代码计算测试集的 TFIDF 指标。但除此之外，我们还要整理数据，比如，为了保证自变量与模型中用到的自变量保持一致，首先要删除一些新词（语料库中没有出现，测试数据集中出现的词）；其次需要给测试数据集补充一些缺失词（测试数据集中没出现，语料库中出现并且用于建模的词）。

- 测试集数据调整

```

1 testtfidf <- testtfidf[!is.na(testtfidf$tfidf),]
2 temp <- unique(testtfidf$term)
3 addterm <- unique(traintfidf$term)
4 addterm <- addterm[!addterm %in% temp]
5 n <- length(addterm)
6 temp <- rep(NA, n*length(testtfidf))
7 temp <- data.frame(matrix(temp, nrow = n))
8 temp[, 3] <- addterm
9 names(temp) <- names(testtfidf)
10 testtfidf <- rbind(testtfidf, temp)
11 tail(testtfidf) #检查一下是否整理正确
12 test <- dcast(data = testtfidf, id + label ~ term, sum, value.var = "tfidf")

```

在上面的代码中，第 1 行代码去除测试数据集中的新词，其特征是 TFIDF 指标为默认值；然后添加测试数据集中缺失的词。第 2 行代码找出测试数据集的词汇。第 3 行代码找出参与建模的所有词汇。第 4 行代码通过 %in% 函数在 temp 中查找参与建模的词汇，使用“非”函数找出那些查不到的词汇。第 5~9 行代码形成一个和 testtfidf 列名相同且等宽，与 addterm 等长的数据框，并且将 addterm 赋值给该数据的 term 列。第 10 行代码将缺失词与测试数据集按行合并。第 11 行代码将数据转化为模型要求的数据框，即将 long 型转为 wide 型，每个词作为一个变量。

- 随机森林测试

```

1 test <- test[!is.na(test$id), ]
2 row.names(test) <- test[, "id"]
3 test <- subset(test, select = -id)

```

```
4 prediction <- predict(Randommodel100, subset(test, select = - label))
5 prediction <- data.frame(cbind(subset(test, select = label),prediction,
row.names(test)))
```

在 R 语言中，测试模型使用 predict 函数，其他特殊的预测函数基本是从这个函数继承过去的方法，只需要输入的测试数据集中的数据和建模数据的变量一样就可以了（顺序不必保持一致）。因为上面补充缺失词的过程中添加了一条 id 为 NA 的记录，所以上面的第 1 行代码删除这条记录。第 2~3 行代码为了删除多余的 id 列。第 4 行代码为测试数据集，subset 函数将测试数据集的因变量列删除。第 5 行代码将预测结果、原标签列和文档 id 按列捆绑，便于下一步统计准确率。

### • 随机森林测试结果

```
1 evalue <- table(prediction$label, prediction$prediction)
2 print(evalue)
#      -1      1
# -1 1836  129
#  1   618 1324
```

测试数据集分类的准确率下降到了 81%，尽管只检验了一次，但是很明显发生了过拟合。此种检验评估的方式实在不符合统计学的思想，也许在这个测试数据集中准确率达到 81%，说不一定在真实数据集中就可能下降到 70%，所以要进一步评估模型准确率的稳定性。

## 10.4.4 算法评估：随机森林应该建多少棵树

随机森林应该构建多少棵树？恐怕很多人都思考过这个问题。在网上搜索一下很难找到特别明确的答案，有的人认为这和数据集的大小有关。一般而言，评价一个事物自然要构建可行的评价指标体系，比如，评价事物的大小时使用均值、中位数等。我们之所以要探究随机森林要构建多少棵树合适，其根本原因是担心树的多少可能影响模型的准确性和稳定性（当然，树枝也可能有影响），那么就需要构建一个评价指标来衡量，比如选择准确性（也可以使用机器学习的方差（variance）和偏差（bias）的概念来评估），那么问题来了，既然进行“比较”，就需要研究比较结果具有的统计学意义。

要符合统计学思想，可以简单地比较正确率的均值和方差来衡量分类的准确性和稳定性。其实这类指标只是看起来老套，但是并不“简单”。我认为在监控、评估监督模型时还是使用一些传统指标比较靠谱，例如，平均绝对误差（MAE）、平均平方差（MSE）、标准平均方差（NMSE）和均值等，它们计算简单、容易理解；只有在非监督模型中才会选择一些所谓“高大上”的指标如信息熵、复杂度和基尼值等。

### 1. $k$ 层交叉检验： $k$ 值为多少时为妙

如果要计算准确率的均值和方差，就需要使用多个测试数据集测试模型，那么，此时只好使用  $k$  层交叉检验（K-fold cross-validation）了。CV 将原始数据随机分成  $k$  组（一般是均分），将其中一个子集作为测试数据集，其余的  $k-1$  组子集作为训练数据集，以此重复  $k$  次，这样会得到  $k$  个模型，

用这  $k$  个模型在  $k$  个测试集上的准确率（或其他评价指标）的平均数作为模型的性能评价指标。

比如，如果要测试 100 棵树和 150 棵树的随机森林模型哪个性能更好，就需要将两个特定参数的模型通过  $k$  层交叉检验，分别构建  $k$  次模型，测试  $k$  次，然后比较它们的均值、方差等指标。那么问题来了， $k$  值应该为多少呢？我可以告诉你，在这方面真的没有“银弹”（Silver Bullet），除了数据集大小的限制，一般来说， $k$  值越小，训练压力越小，模型方差越小，而模型的偏差越大； $k$  值越大，训练压力越大，模型方差越大，而模型偏差越小。

进行交叉检验首先要对数据分组，数据分组要符合随机且平均的原则。遵循这两条原则可以写一个交叉检验分组的自编函数：CVgroup，它包含 3 个参数：cross 用于设定  $k$  值，即准备做几层交叉检验。datasize 用于设定数据集大小，即有多少条观测值，简单理解就是数据框有多少行。seed 用来设置随机种子，所谓随机种子，如果设定了这个值，即可保证现在生成的随机数和你要生成的随机数是相同的，虽然在程序中有随机过程，但是可以保证你在验证别人的程序时，不会因为随机过程而改变结果，换句话说，大家对随机过程使用了同一套锁钥。

这里没有使用现有的包进行  $k$  层交叉检验分组，因为，我发现，目前已经公布的方案大多不能平衡“均分”这个原则（虽然不是必需的），所以自编分组函数解决问题。鉴于大家需要锻炼根据伪代码写函数的能力，所以下面将代码尽量拆分开说明。分组的思路是这样的：可以先产生一个  $1:k$  的序列，然后将这个序列复制成和数据编号长度（datasize）相等或略大于数据编号长度，从前往后依次抽出和数据编号等长的序列，为序列  $n$ 。然后从  $n$  中非放回性随机抽样，抽出一个和数据编号长度相等的序列 temp，这时 temp 随机地包含  $1:k$  序列中的任何一个值且它们的个数基本相等，更重要的是，它和数据编号等长。然后按照每一个  $k$  值筛选数据编号并分为一组就可以了。

```
1 library(plyr)
2 CVgroup <- function(k, datasize, seed) {
3   cvlist <- list()
4   set.seed(seed)
5   n <- rep(1:k, ceiling(datasize/k))[1:datasize]
6   temp <- sample(n, datasize)
7   x <- 1:k
8   dataseq <- 1:datasize
9   cvlist <- lapply(x, function(x) dataseq[temp==x])
10  return(cvlist)
11 }
12 k <- 5
13 datasize <- nrow(iris)
14 cvlist <- CVgroup(k = k, datasize = datasize, seed = 1206)
```

在上面代码中，第 3 行产生一个空的 list，用于盛装每一个小组的观测值编码（或指行编号）。第 4 行代码用于设置随机种子。第 5 行代码中的 ceiling 函数用于向上取顶，比如 1.2 取顶就是 2；对应的函数有 floor，向下取低，取顶后用于设定序列  $1:k$  复制的次数。因为要保证复制（rep）后的长度略大于数据大小，所以向上取顶。然后依次从中抽出前  $1:datasize$  个数据，形成一个和数据等长的序列  $n$ ，这样做的目的是尽量保证均分的原则。第 6 行为非放回随机抽样，抽取和数据等长的序列

temp，这句代码的意思很简单，就是将原来的数据列随机重新摆列，保证遵循随机的原则。第 7 行代码生成一个用于筛选的序列  $1:k$ 。第 8 行代码生成数据编号 dataseq。第 9 行代码因为 temp 序列和 dataseq 序列一一对应，所以以 temp 筛选 dataseq 其结果应该也是随机的，这里使用 lapply 函数将每一个  $x$  值用于匿名函数。匿名函数是一个简单的筛选，向量筛选和数据框不同，数据框筛选需要有表示行和列分界的英文逗号“，”，如 `df[temp==x,]`，而向量则不用“，”分割，因为向量可以被视为一个只有一列的矩阵。

- $k$  层交叉检验数据集

最后 3 行代码用于对著名的 iris 鸢尾花数据集分组，设定  $k$  值。nrow 函数用于获得鸢尾花数据集有多少行，即 `datasize`。最后一行代码使用函数对数据编号分组。

以上是对非序列数据的交叉检验分组。那么问题来了，如果是序列性的数据（比如时间序列），那么怎么分组呢？可以以测试数据集的大小为步长向前滑动。

下面就验证一下随机森林的树数对模型准确性的影响。首先要确定树的序列，比如要测试 60~200 棵树，可以每隔 20 棵设定一个特定的树数（根据数据而定），然后对每种设定树数的随机森林做一次  $k$  层检验，这样就会得到  $k$  个预测结果集。如果测试  $n$  种树数，那么就要得到  $k \times n$  个预测结果数据集。

按照上面的结果，我们需要进行一个嵌套循环。循环的外层是任何一种设定的树数，比如，对 60 棵树循环一次，对 80 棵树也要循环一次等，依次循环到  $n$ 。循环的内层针对某一设定树数的随机森林进行  $k$  层交叉检验循环，比如，使用分组 1 和其余  $k-1$  组数据对 60 棵树的随机森林进行一次建模和测试，然后使用分组 2 和其余  $k-1$  组数据对 60 棵树的随机森林再做一次建模和测试，依次循环到  $k$ 。

这里的数据集不再使用上面的情感分析数据集测试，而是使用 R 自带的鸢尾花数据集，主要是为了节省建模时间。如果使用情感分析数据集，则估计一般的计算机很难承受计算量，即使放在服务器上也要花两天时间才能分析完。这里使用鸢尾花数据集预测鸢尾花的萼片长度，萼片长度为因变量，数据集中其他变量为自变量，这一任务的性质为回归的预测而非分类。

- 随机森林  $k$  层较差检验

```
1 data <- iris
2 pred <- data.frame() #存储预测结果
3 library(plyr)
4 library(randomForest)
5 m <- seq(60, 500, by = 20)##如果数据量大，则尽量间隔大一点，间隔过小没有实际意义
6 for (j in m) {
7   progress.bar <- create_progress_bar("text")
8   progress.bar$init(k)
9   for (i in 1:k) {
10    train <- data[-cvlist[[i]],]
11    test <- data[cvlist[[i]],]
12    model <- randomForest(Sepal.Length ~ ., data = train, ntree = j)
13    prediction <- predict(model, subset(test, select = - Sepal.Length))
```

```

14  randomtree <- rep(j, length(prediction))
15  kcross <- rep(i, length(prediction))
16  temp <- data.frame(cbind(subset(test, select = Sepal.Length), prediction,
randomtree, kcross))
17  pred <- rbind(pred, temp)
18  print(paste("随机森林: ", j))
19  progress.bar$step()
20  }
21  }

```

在上面的代码中，第 1 行代码为创建数据集。第 2 行为创建一个空数据框，用于存放测试数据集编号、树数、真实值和预测值。第 5 行代码生成一个包含欲测试树数的向量，这里测试 60~200 棵树。第 6 行为第一层循环，每种树数循环一次。第 7~8 行代码用 `plyr` 包的 `create_progress_bar` 函数创建一个进度条，第 8 行代码设置任务数，比如，下面的循环要进行  $k$  次，则任务数为  $k$ 。第 9 行代码为内层循环开始，即  $k$  层交叉检验。第 10 行代码为筛选 `data` 中行编号与交叉检验分组 `cvlist` 里第  $i$  个元素存储的行编号相同的数据，将其删除后余下的数据（其余  $k-1$  组）作为训练数据集，而筛选行编号相同的数据作为测试数据集（第 11 行代码）。第 12 行代码使用训练数据集构建随机森林模型，树数为  $j$ 。第 13 行代码使用上一步构建的模型预测测试数据集的萼片长度，`subset` 函数删除了测试数据集中真实的萼片长度。`subset` 函数非常有用，能够让我们非常方便地按照列名称筛选、提取、删除数据列，而不是使用列编号，在写代码时尽量不要使用编号筛选。第 14 行代码将树数  $j$  重复至与预测值等长，赋值给 `randomtree`。第 15 行代码将  $i$  重复至与预测值等长，赋值给 `kcross`。第 16 行代码将真实值、预测值、随机森林树数、测试组编号捆绑在一起组成新的数据框 `temp`。第 17 行代码将 `temp` 按行和 `pred` 合并。第 18 行代码打印通知：循环至树数  $j$  的随机森林模型。第 19 行代码输出进度条，告知完成了这个任务的百分之几。这样我们就可以根据 `pred` 记录的结果进行方差分析等，进一步研究树数对随机森林准确性及稳定性的影响。

上面的循环也可以使用 `lapply` 家族函数进行改造，甚至使用并行计算都可以，你可以尝试一下。相关代码可以参看本书的资源下载文件。

- `lapply` 家族提高效率

```

1  data <- iris
2  library(plyr)
3  library(randomForest)
4  j <- seq(10, 10000, by = 20)
5  i <- 1:k
6  i <- rep(i, times = length(j))
7  j <- rep(j, each = 5)
8  x <- cbind(i, j)
9  cvtest <- function(i, j) {
10   train <- data[-cvlist[[i]],]
11   test <- data[cvlist[[i]],]
12   model <- randomForest(Sepal.Length ~ ., data = train, ntree = j)

```

```
13 prediction <- predict(model, subset(test, select = - Sepal.Length))
14 temp <- data.frame(cbind(subset(test, select = Sepal.Length),
prediction))
15 }
16 system.time(pred <- mdply(x, cvtest))
```

当测试的循环数较少或单任务耗时较少时，apply 函数家族并不比循环具有效率上的优势，但一旦比赛由百米赛跑变成了马拉松，apply 函数家族的优势就展现出来了，这就是所谓的路遥知马力吧。

## 2. 模型评估：随机森林应该有多少棵树

既然是预测回归任务，就可以使用常见的统计指标进行模型的筛选、评估、监控。这里仅回顾平均绝对误差（MAE）、均方差（MSE）和标准化平均绝对方差（NMSE）这 3 个评价指标，其计算方式如下：

- 统计指标

```
平均绝对误差 = mean(abs(预测值-观测值))
均方差 = mean((预测值-观测值)^2)
标准化平均方差 = mean((预测值-观测值)^2)/mean((mean(观测值) - 观测值)^2)
```

这三者各有优、缺点，就单个模型而言，虽然平均绝对误差能够获得一个评价价值，但是你不知道这个值代表模型拟合是优还是劣，只有通过对比才能知道效果。均方差也有同样的毛病，而且均方差由于进行了平方运算，所得值的单位和原预测值不统一了，比如，观测值的单位为米，均方差的单位就变成了平方米，更加难以比较。标准化平均方差对均方差进行了标准化改进，通过计算拟评估模型与以均值为基础的模型之间准确性的比率。标准化平均方差取值范围通常为 0~1，比率越小，说明模型越优于以均值进行预测的策略，标准化平均方差的值大于 1，意味着模型预测还不如简单地把所有观测值的平均值作为预测值，但是通过这个指标很难估计预测值和观测值的差距，因为它的单位也和原变量不一样了。综合各个指标的优、缺点，我们使用这 3 个指标对模型进行评估。

- 回归模型的筛选指标

```
1 maefun <- function(pred, obs) mean(abs(pred - obs))
2 msefun <- function(pred, obs) mean((pred - obs)^2)
3 nmsefun <- function(pred, obs) mean((pred - obs)^2)/mean((mean(obs) -
obs)^2)
4 library(dplyr)
5 eval <- pred %>% group_by(randomtree, kcross) %>%
6   summarise(mae = maefun(prediction, Sepal.Length),
7             mse = msefun(prediction, Sepal.Length),
8             nmse = nmsefun(prediction, Sepal.Length))
```

前 3 行代码分别构建了计算 mae、mse、nmse 的 3 个函数。从第 4 行代码开始使用 dplyr 包里的管道函数“%>%”将数据集传递给 group\_by 函数，按照随机森林的树数 randomtree、交差检验测试集编号 kcross 这两列进行数据分组，然后使用 summarise 函数将数据集中相应的变量应用到 3 个指标计算函数，最后得到了一个包含每一种特定树数的随机森林在特定测试数据集上的 3 个评价指标集。

检验不同树数的随机森林的 3 个指标是否存在显著的差异，其实就是进行单因子方差分析。在



进行方差分析之前，首先要检验方差齐性，因为在方差分析的  $F$  检验中，是以各个实验组内总体方差齐性为前提的；方差齐性通过后进行方差分析，如果组间差异显著，再通过多重比较找出哪些组之间存在差异。

- 单因子方差分析

```
1 eval$randomtree <- as.factor(eval$randomtree)
2 bartlett.test(mae ~ randomtree, data = eval)
3 temp <- aov(mae ~ randomtree, data = eval)##可以选择前 100 行
4 summary(temp)
5 TukeyHSD(temp)
#   Tukey multiple comparisons of means
#     95% family-wise confidence level
#
# Fit: aov(formula = mae ~ randomtree, data = eval[1:100, ])
#
# $randomtree
#           diff          lwr          upr      p adj
# 30-10    0.0350503857 -0.1566504 0.2267512 1.0000000
# 50-10   -0.0067381040 -0.1984389 0.1849627 1.0000000
# 70-10    0.0392858727 -0.1524149 0.2309867 0.9999997
# 90-10   -0.0099382978 -0.2016391 0.1817625 1.0000000
```

在上面的代码中，第 1 行代码首先要将分组变量转化为因子。第 2 行代码使用 bartlett 方法检验指标 mae 的方差齐性。为什么检验方差齐性？其目的是保证各组的分布一致，如果各组的分布都不一致，那么比较均值还有什么意义？ $F$  值越小（ $p$  值越大，大于 0.05），就证明没有差异，说明方差齐；aov 函数对 MAE 指标进行方差分析，summary 显示差异不显著，说明不同树数的随机森林的 MAE 指标差异不显著（ $p$  值远远大于 0.05），即没有必要做多重检验了。但为了展示整个分析流程，我们使用 TukeyHSD 进行多重比较，各组间的  $p$  adj 值都远远大于 0.05，组间不存在差异，以上就是分析评估代码，如果你感兴趣的话，则可以稍作修改进一步检验 MSE、NMSE 这两个指标。

统计检验让我们坚信各种树数的随机森林之间的差异不显著，但是很多人总是坚信眼见为实，下面将 3 个指标随树数的变化趋势可视化，使用折线图分析一下它们的差异。

- 自备绘图函数

```
1 title_with_subtitle = function(title, subtitle = "") {
2   ggtitle(bquote(atop(.(title), atop(.(subtitle)))))
3 }
4 library(extrafont)
5 loadfonts(device="win")
```

上面加载添加主副标题的函数，同时加载了 Windows 系统自带的字体，以丰富绘图时可以使用字体类型，在绘制折线图之前首先要对数据进行简短的调整。

- 折线图分析

```
1 library(ggplot2)
2 library(reshape2)
```

```

3 eval <- aggregate(cbind(mae, mse, nmse) ~ randomtree, data = eval, mean)
4 eval <- melt(eval, id = "randomtree")
5 eval$randomtree <- as.numeric(as.character(eval$randomtree))
6 p <- ggplot(eval, aes(x = randomtree, y = value, color = variable)) +
7   geom_line(size = 1.3) +
8   geom_vline(xintercept = 250, color = "#FF1493", size = 1.3) +
9   facet_wrap(~ variable, nrow = 3, scales = "free") +
10  scale_color_manual(values = c("#800080", "#FF6347", "#008B8B")) +
11  #scale_x_continuous(breaks = seq(0, 10000, by = 100)) +
12  ylab("") +
13  xlab("随机森林树数") +
14  title_with_subtitle("随机森林应该有多少棵树", "抗过拟合并非不会过拟合") +
15  theme_bw(18) + theme(panel.background = element_rect(fill = rgb(red = 242,
242, max = 255)),
16                        plot.background = element_rect(fill = rgb(red = 242,
                                                                    green = 242,
blue = 242, max = 255)),
17                        plot.title = element_text(size = rel(1.2), family =
"STXingkai", face = "bold", hjust = 0.5, colour = "#3B3B3B"),
18                        panel.grid.major = element_line(colour=rgb(red = 146, green
= 146, blue = 146, max = 255),size=.75),
19                        panel.border = element_rect(colour = rgb(red = 242,
                                                                    green = 242, blue
= 242, max = 255)),
20                        axis.ticks = element_blank(),
21                        axis.text.x = element_text(colour = "grey20", size = 8),
22                        axis.text.y = element_text(colour = "grey20", size = 10),
23                        axis.title.y = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = 0.5),
24                        axis.title.x = element_text(size = 11, colour = rgb(red =
74,green = 69, blue = 42, max = 255), face = "bold", vjust = -0.5),
25                        legend.background = element_rect(fill = rgb(red = 242,green
= 242, blue = 242, max = 255)),
                        legend.position = "")

```

在上面的代码中，第 3 行代码按树数对 3 个指标做透视表求取均值。第 4 行代码中的 melt 函数将数据表从 wide 型转换为 long 型，便于使用 ggplot2 作图。第 6 行代码用于将原来为整数类型变量转换为因子变量，便于 ggplot2 按照因子水平分组（见图 10-1）。

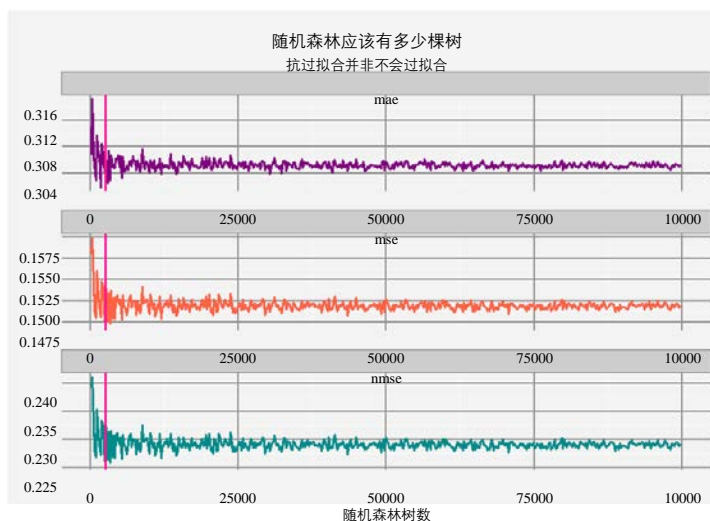


图 10-1

## 3. 说一说基于算法的情感分析的坏话

说实话，基于算法的情感分析方法要比基于词典的情感分析方法的准确性好很多，虽然我们只是简单地使用 DF 指标构建随机森林模型，其效果可谓立竿见影，但基于算法的情感分析方法真的要一统天下了吗？

首先，就准确率而言，基于算法的情感分析方法还有待提高，而目前的算法模型准确性很难再上一个层次，所以，研究者要么创造更新、更强大的算法，要么转向寻求其他的解决方案，以使准确率更上一个台阶。

其次，由于上述方法使用了所有词汇，并没有进行特征词筛选，这样会给机器造成很大的压力。如果文本越来越多，词汇变量也会越来越多，矩阵越来越稀疏，计算量越来越大，这样在挑选算法的同时，我们将不得不解决另外一个问题，即特征词的提取。这里的特征词提取方法不是一般的特征词提取方法就能解决的，其目的是提取能够区分情感倾向的特征词，所以，找到能够实现目的的方法也着实不易。

最后，基于算法的情感分析方法一般具有行业特殊性，也就是说很难训练一个可以跨行业的模型，这样就会遇到另外一个问题：挑选训练样本。比如，将本来是针对汽车销售行业构建的模型迁移到快消行业，准确性就有可能下降。为了保证准确性，需要挑选快消行业的训练数据集进行重训练，那么问题来了，这种训练数据集一般要成千上万条文本评论，人工挑选的话也许会让人筋疲力尽，两眼发黑。

所以在我看来，目前以上 3 点是基于算法的情感分析方法需要改进和提高的关键点，至于分析情感的细腻程度、情感主体归属等问题就不仅仅是基础算法这一种方法存在的问题了，其他方法同样也会遇到这类麻烦，可以另外作为一个新的课题进行研究。

## 10.5 一种准确率高达 90%的新方法

以上探讨了两种常见的情感分析方法，同时分析了二者的优劣势。很明显，基于词典的方法虽然找准了特征词（人工整理情感词），但其准确性确实是无法逾越的鸿沟；而基于算法的分析方法需要人工挑选大量的训练数据集，同时还要涉及优秀的特征词提取方法以减轻分析压力。虽然现在有一些研究者提出了一些奇思妙想，比如，根据文本中的表情符号自动将文本归属到某一情感类别，虽然比较武断，但也确实减轻了人工挑选训练数据集的工作压力。这样无论是自动提取情感词，还是做情感分析，都有了可以想象的空间，比如，可以根据某些词在正、负样本中的概率分布的差异自动提取情感词，也可以使用大量的标注样本训练模型，所以，解决人工标注训练库问题是基础问题。

### 10.5.1 拿来主义的启示

下面介绍一种新方法。说是新方法其实也不新，只不过我们在这里大胆地使用“拿来主义”：既然算法的分析方法准确率比较高，那么一定要把算法包含到新方法中。但是算法需要很好的特征词库作为基础，以减轻计算压力。我们不是有大量的情感词库吗？这些情感库里的词语已经足够丰富了，完全可以作为特征词。在情感词库中直接将词分为正向情感词和负向情感词，这样做比较武断，那么我们可以不对情感词进行区分，统计一个情感词在正向文档中的 DF 指标，同时统计它在负向文档中的 DF 指标，并分别作为词汇的情感权重，这样就轻松解决了情感词权重的问题。那么问题来了，怎么挑选训练数据集？如果我们的情感分析引擎具有跨行业分析的功能，就不用挑选训练数据集了，基于这一点仍然使用“拿来主义”，即将网络上其他人标注的训练数据集全部拿来，这样我们的训练数据集就有可能横跨好几个行业了。一旦模型具有跨行业分析的能力，就可以省去在本行业的数据里标注训练数据集的烦琐工作了。但是，有人仍然会怀疑网络上的数据集比较小，不能保证跨所有行业。没关系，只要你同意这样一个假设就好了。在一般情况下，人类描述情感的词汇在各行各业是通用的，比如“喜欢”“讨厌”“告状”“恶俗”“次品”等，也就是说，输入模型的特征变量在各行各业都是一样的，而且意义也基本相似，这就更进一步保证了模型跨行业分析的效果。

#### ① 重要假设

在绝大多数情况下，无论哪个行业，人类描述情感的词汇基本是相同的。

同时，由于情感词有了权重，我们的模型就可以将分类任务转化为更加细腻的情感分析方式，比如将概率转化为得分等。另外，算法有时候不能高效解决否定词出现的情况，比如“我不喜欢你”，还有出现程度副词的情况，比如“非常喜欢你”等。其实可以适当添加几个规则，使用规则来改变情感词的权重就可以了。

#### ② 新方法涉及内容

- (1) 算法：保证情感分析的准确率。
- (2) 情感词库：情感词作为特征词，解决特征词提取的问题、突破行业限制。
- (3) TFIDF 加权：使用 DF 指标作为情感词添加正负向权重，保证情感分析更加细腻。

- (4) 训练库：使用跨行业的训练数据集迫使模型突破行业限制。
- (5) 否定词规则：提高算法分析否定句的能力，保证情感分析的细腻性。
- (6) 程度副词规则：保证情感分析的细腻性。

我给这个新的混合规则取了一个比较拗口的名字：基于加权词典规则的半监督式情感分析。当然如果你觉得好用，也可以称之为“要你命 3000”。

## 10.5.2 情感词典和规则构建

通过以上分析，要完成这样一个情感分析引擎，至少要经过以下 4 个步骤。

- (1) 制作加权的情感词库（特征词）。
- (2) 添加否定词规则，比如情感词前面出现否定词应该怎么调整。
- (3) 添加程度副词规则，比如情感词的前后出现程度副词应该怎么调整。
- (4) 选择合适的算法。

以上几个步骤和问题想通了，接下来的处理过程就比较顺畅了。

### 1. 计算情感词 TFIDF 权重

前面已经说了，我们将情感词作为特征词，然后转化为模型的输入变量。首先要给情感词一定的权重，而不是武断地将情感词分为负向情感词或正向情感词，相反，我们将通过数据来定义一个情感词的权重。

- 定义情感词的权重

将一个情感词在正向文本集出现的文档频率（DF）作为它的正向权重，而在负向文本集出现的文档频率作为它的负向权重。

换句话说就是，计算一个情感词在负向文本集中出现的概率作为它的负向权重，同时计算它在正向文本集中出现的概率作为它的正向权重。比如“喜欢”这个词一般情况下在正向文本集中出现的概率大于在负向文本集中出现的概率。依据这个定义来整理情感词库，因为在前面我们已经整理了情感词库和正、负向文本集，因此只需要分别统计情感词在正向文本集中出现的 DF 指标和在负向文本集中出现的 DF 指标即可。

- 整合词典

```
1 pos <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/
自备词典/pos.csv", header = T,
                sep = ",", stringsAsFactors = F)
2 type <- rep(1, length(pos[,1]))
3 pos <- cbind(pos, type)
4 neg <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/
自备词典/neg.csv", header = T,
                sep = ",", stringsAsFactors = F)
5 type <- rep(-1, length(neg[,1]))
6 neg <- cbind(neg, type)
```



```
8 names(posdf) <- c("term", "pdf")
```

以上代码计算了正向样本中词汇的 PDF（加 P 表示正向文本 positive）。其中第 1 行代码去除个别词中含有的空格，便于后面的计算，例如“我们”和“我们”。第 2 行代码提取非空数据。第 3 行代码筛选出情感词汇。第 4 行代码去重，计算 DF 指标（需要对每篇文章的词汇去重）。第 5 行代码提取正向的文本。第 6 行代码计算总共有多少篇正向文章。第 7 行代码计算 DF 指标。第 8 行代码重命名。

- 情感词添加正向权重

```
1 library(plyr)
2 tvpn <- join(posdf, posneg, type = "full")
3 tvpn <- unique(tvpn)
```

在上面的代码中，第 2 行代码使用 join 函数将 posdf 和 posneg 数据框按照 term 列关联在一起，关联类型使用全关联。然后使用 unique 函数去重即可，使用 PDF 指标作为情感词的正向权重，任务就完成了，如果你发现 DF 指标的计算结果和原定义不太一样，没关系，这样计算反而能提高准确率。

## （2）计算负向文本集中情感词 DF 指标的权重

完成情感词库的正向 DF 指标加权后，就要做负向文本 DF 指标加权，其过程和之前的操作一样。

- 负向文本 DF 指标加权

```
1 negtrainterm <- trainterm[trainterm$label == -1, ]
2 total <- length(unique(negtrainterm$id))
3 negdf <- data.frame(table(negtrainterm$term)/total)
4 names(negdf) <- c("term", "ndf")
```

- 情感词添加负向权重

```
1 temp <- join(negdf, posneg, type = "full")
2 temp <- unique(temp)
3 tvpn <- join(tvpn, temp, type = "full")
```

代码和上面的相似，只是改了一下列名称而已（n 表示负向文本集），最后使用 join 函数给情感词添加负向权重，这样大多数情感词具有了以数据驱动的情感权重。但是，还有很多词，要不缺少负向权重，要么缺少正向权重，或者干脆什么都没有。也就是说由于训练数据集太少，没有囊括所有的情感词。这样就需要设计缺失值填充的规则，一般缺失值填充可以使用均值、中位数、相关性拟合预测等，但这里需要设计一套相对合理的方案，当然你也可以尝试其他方法。

- 填充规则 1

PDF 指标和 NDF 指标同时缺失，即训练数据集中未出现该词：

如果情感词人工分类的 type 分类为 1，即为正向，则 PDF 指标用 PDF 指标的 median（中位数）填充，NDF 指标用 NDF 指标的 min（最小值）填充。

如果情感词人工分类的 type 分类为 -1，即为负向，则 PDF 指标用 PDF 指标的 min（最小值）填充，NDF 指标用 NDF 指标的 median（中位数）填充。

例如：

```
1 tempno <- tvpn[is.na(tvpn$pdf) & is.na(tvpn$ndf), ]
2 tempno[which(tempno$type == 1), "pdf"] <- median(tvpn$pdf[!is.na
```

```
(tvpn$pdf))
3 tempno[which(tempno$type == 1), "ndf"] <- min(tvpn$ndf[!is.na(tvpn$ndf)])
4 tempno[which(tempno$type == -1), "pdf"] <- min(tvpn$pdf[!is.na(tvpn$pdf)])
5 tempno[which(tempno$type == -1), "ndf"] <- median(tvpn$ndf[!is.na(tvpn$ndf)])
```

在上面的代码中，第 1 行代码使用 & ( 且 ) 函数，筛选出训练数据集中未出现的词汇。按照填充规则 1，第 2 行代码筛出 type 为 1 的词汇并将 tvpn 中 PDF 指标的中位数赋值给相应的 PDF 指标缺失值。第 3 行代码筛出 type 为 1 的词汇并将 tvpn 中 NDF 指标的最小值赋值给相应的 NDF 指标缺失值。第 4 行代码筛出 type 为 -1 的词汇并将 tvpn 中 PDF 指标的最小值赋值给相应的 PDF 指标缺失值。第 5 行代码筛出 type 为 -1 的词汇并将 tvpn 中 NDF 指标的中位数赋值给相应的 NDF 指标缺失值。

- 填充规则 2

PDF 指标和 NDF 指标仅缺失一项时：

如果情感词缺失 PDF 指标权重，即未在正向文本中出现过，则 PDF 指标用 PDF 指标的 min( 最小值 ) 填充。

如果情感词缺失 NDF 指标权重，即未在负向文本中出现过，则 NDF 指标用 NDF 指标的 min( 最小值 ) 填充。

例如：

```
1 temp <- tvpn[!(is.na(tvpn$pdf) & is.na(tvpn$ndf)), ]
2 temp[is.na(temp$pdf), "pdf"] <- min(tvpn$pdf[!is.na(tvpn$pdf)])
3 temp[is.na(temp$ndf), "ndf"] <- min(tvpn$ndf[!is.na(tvpn$ndf)])
4 tvpn <- unique(rbind(tempno, temp))
5 write.csv(tvpn, 'H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/
自备词典/tvpn.csv', row.names = FALSE)
```

在上面的代码中，第 1 行代码筛出两种指标非完全缺失的情况。第 2 行代码对于缺失 PDF 的情况，使用 tvpn 中 PDF 指标最小值填充 ( 移除了缺失值 )。第 3 行代码对于缺失 NDF 指标的情况，使用 tvpn 中 NDF 指标最小值填充。第 4 行代码将填充完成的数据集重新合并在一起，并去除重复的数据 ( 也许是多余的，但以防万一 )。这样就有了一部数据加权的情感词典，然后就可以使用权重测试各类算法的分类效果了。

其实数据挖掘并不只是各类代码的实现。一项数据挖掘任务集中了各种零散的知识，其中任何一个过程都需要统计学作为支撑。想当然往往害人不浅，比如，我们可能认为分数为四颗星的评论可以被归为正向评论，但你是否发现某些电商网站的商品评分都在四颗星以上，这时就需要使用样本分布来界定了。

除需要具有严密的统计学知识外，还需要具有将零散的东西组装为一个系统的能力，比如，一个任务使用 K-Means 算法进行聚类、使用轮廓系数评估聚类数多少的问题、使用因子转换的方式解释类的特征、使用三维图展示聚类结果，思维的连贯性要与任务的连贯性不谋而合。



## 2. 情感规则构建

有了加权词典，然后需要建立几个规则。规则具有普遍性，在极端情况下，规则可能并不适用，但是在大多数情况下，规则都能够发挥作用，这就是我们建立规则的共识。仔细观察评论文本或我们的日常用语习惯，至少可以构建两类规则：（1）程度副词规则，即在情感词的前后出现程度副词，比如“非常喜欢”或“可恶至极”等。程度副词起到了加强或弱化情感词的作用。（2）否定词逆转规则，即在情感词的前面出现否定词，比如“我不喜欢”，这样“喜欢”这个词的词性就被逆转了。

基于以上规则，下面先建立程度副词规则。因为大多数程度副词具有加强作用（其实是为了简单明了），所以，这里武断地认为所有的程度副词都是将原有的情感加强为两倍，当然，你也可以给每一个程度副词人工附上一个加强或削弱的权重（如果你对下面的程序运行结果不满意，那么这可能是一个提升分析引擎准确性的方法）。

根据规则，调整完数据后，下面使用前面写的简单的朴素贝叶斯算法计算文本的正、负向得分，即将一个文本中情感词的正向权重累乘结果作为该文本的正向得分，负向权重累乘结果作为它的负向得分，如果正向得分大于负向得分，则该文本的情感为正。没错，这里好像有很多不符合朴素贝叶斯原理的地方，我不避讳这一点，随后会有很多违反统计规则的地方，因为我们的原则是：做项目不像做研究，做项目的目的是让其尽快落地。

### • 程度副词规则

如果在情感词前后不远处出现了任意一个程度副词，那么在该情感词的正、负权重中，较大者加倍（且仅加一次），例如“我非常喜欢喝茶”，“喜欢”这个词的正向情感权重为 0.05，负向情感权重为 0.02，它的前面出现了“非常”程度副词，所以“喜欢”的正、负向权重就分别变成了 0.1 和 0.02。

因为朴素贝叶斯算法的分类过程已经指定了，所以这里直接使用测试数据集测试即可。在输入模型之前需要按照设计的规则调整权重，比如上述的程度副词规则，在调整之前还是要先完成分词整理的。

### • 测试数据集分词整理

```
1 test <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/test.csv", sep = ",", header = T, stringsAsFactors = F)
2 sentence <- as.vector(test$msg)
3 sentence <- gsub("[[:digit:]]*", "", sentence) #清除数字[a-zA-Z]
4 sentence <- gsub("[a-zA-Z]", "", sentence)
5 sentence <- gsub("\\\\.*", "", sentence)
6 test <- test[!is.na(sentence), ]
7 sentence <- sentence[!is.na(sentence)]
8 test <- test[!nchar(sentence) < 2, ]
9 sentence <- sentence[!nchar(sentence) < 2]
10 insertWords(dict)
11 system.time(x <- segmentCN(strwords = sentence))
12 temp <- lapply(x, length)
13 temp <- unlist(temp)
14 id <- rep(test[, "id"], temp)
```

```
15 label <- rep(test[, "label"], temp)
16 term <- unlist(x)
17 testterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)
18 testterm <- testterm[testterm$term %in% tvpn$term, ]
19 testterm <- join(testterm, test[, c("id", "msg")])
```

上面的分词过程和前面一样，这里就不再多说了。这里仅添加了最后两行代码，倒数第二代码行使用 %in% 函数筛选出情感词，最后一行代码使用 join 函数按照 id 匹配 msg 列。如果觉得代码比较啰嗦，则可以写一个复杂一点的函数，并将其打包为函数或脚本。

在按照程度副词规则调整情感词的权重之前，首先要介绍一下具体的实现方法：将原文按照情感词切分为  $n$  个小片段，然后在这些小片段的特定位置查询是否存在程度副词。另外，我们将程度副词按照位置分为了两类：（1）在情感词之后，比如“之极”“之至”。（2）在情感词之前，例如“很”“非常”。先在情感词之后的片段中的前 4 个字符里搜索第一类程度副词；然后在情感词之前的片段中搜索第二类程度副词。两者的搜索方法不一样，前者使用正则表达式，后者先分词再进行匹配，我认为这两种方法各有优、缺点，好在都不太影响程序运行速度。

### （1）切分句子

```
1 test <- split(testterm, testterm$id)
2 splitsentence <- function(x) {
3   term <- x$term
4   nnum <- nchar(term)
5   term <- term[order(nnum, decreasing = TRUE)]
6   term <- paste(unique(term), collapse = "|")
7   fragment <- strsplit(x$msg[1], term)#如果文本的情感词总数大于 933 将报错，
改用循环切分
8   x$msg <- unlist(fragment)[1:length(x[,1])]
9   return(x)
10 }
11 temp <- lapply(test, splitsentence)
12 fragment <- do.call("rbind", temp)
13 fragment[is.na(fragment$msg), "msg"] <- "a"
```

在上面的代码中，第 1 行代码使用 split 函数将 testterm 数据框按照 id 列分组，然后将数据框转为 list，list 中的每一个元素是 id 相同的原数据框小组（比较像 group\_by）。第 2 行代码为一个不太规范的自编函数，参数 x 的格式要求与上一步产生的数据框小组相同，即与上一步产生的 list 中的元素格式相同，并在 splitsentence 函数中先提取文本中的情感词作为向量，然后使用 nchar 函数计算每一个情感词的字符长度。接着按照字符长度对情感词排序，情感词越长，位置越靠前。看到这里，读者可能感觉到有点儿莫名其妙，为什么要这么做？比如“妹妹很感激刘强，对刘强的帮助感激涕零遂以身相许”其中情感词“感激”在情感词“感激涕零”之前，如果按照原有的顺序切分，就会被切分为“妹妹很”、“刘强，对刘强的帮助”、“涕零遂以身相许”，但这显然不是我们愿意看到的，应该先按照“感激涕零”切分后，再按照“感激”切分，所以需要将较长的短语位置调整到前面来优先切分。其实很多分词算法也有这样的原则，较长的词汇往往权重较高。第 6 行代码将调整后的

情感词粘贴在一起，并用“|”符号分割，“|”符号在正则表达式中表示“或”的关系。第 7 行代码使用 `strsplit` 函数将原文本按照 `term` 切分，只要碰到 `term` 中的任意一个词就切分一次，其实这两句代码也可以写循环语句完成，但这种循环不可并行，上一个词的切分结果将按照下一个词切分。这里使用“或”的关系将它们一次切分，速度比循环语句快，但是也有一个缺点：这种“或”的关系不能超过 900 个词。一般一个评论不会包含 930 多个不重复的情感词，除非你分析的是一本文学著作。

第 8 行代码将切分的结果 `unlist`，然后提取 `1:length(x[,1])` 个片段。因为一个词一般会将一句话分成两段，两个词会将一句话分 3 段，为了片段和词的数量对应，这里只能妥协，舍去了切分结果的最后一个片段，有  $n$  个情感词，就提取切分结果的前  $n$  个片段，最后使 `return` 函数返回 `x`，如果没有 `return` 函数，则 R 将返回函数形成的最后一个对象“`x$msg`”。

然后使用 `lapply` 函数将 `test` 中的每一个元素作为参数 `x` 传给切分函数 `splitsentence`，最后使用 `do.call` 函数将结果数据类型由 `list` 按行捆绑（`rbind`）为数据框。在切分过程中产生了一些缺失值，最后一行代码将这类缺失值替换，这样就基本上将评论原文按照情感词切分成功了。

然后就可以先查找、匹配第一类程度副词，它们“躲”在情感词的后面。规则是这样的：如果在一个片段的前 4 个字符中找到了第一类程度副词，那么该片段前面的情感词的权重就需要调整，即统一将正、负权重中较大者增加一倍。如果你觉得这种方式太武断，那么也可以使用本书提供的加权程度副词词典，给每一种程度副词赋予不同的权重。

#### • 查找第一类程度副词

```
1 afterdegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分
析/dict/自备词典/afterdegree.csv", header = T, sep = ",", stringsAsFactors = F)
2 afterdegree <- paste(afterdegree$msg, collapse = "|")
3 temp <- split(fragment, fragment$id)
4 afterdegreefun <- function(x, afterdegree) {
5   if (length(x$term) > 1) {
6     piece <- substr(x$msg, 1, 4)
7     k <- regexpr(afterdegree, piece)
8     scoreafter <- ifelse(k > 0, 2, 0)
9     scoreafter <- scoreafter[-1]
10    scoreafter <- c(scoreafter, 0)
11  } else scoreafter <- 0
12  return(scoreafter)
13 }
14 temp <- lapply(temp, afterdegreefun, afterdegree)
15 scoreafter <- unlist(temp)
```

在上面的代码中，第 1 行代码读取第一类程度副词。第 2 行代码将所有的程度副词粘贴在一起，并使用“|”符号隔开。在切分阶段，我们将类似的一步操作放在了切分函数 `splitsentence` 里面，因为这里的 `afterdegree` 函数不会随函数参数的变化而变化，如果放在函数外面，则仅执行一次，放在函数里面则每次调用函数时都会执行一次。第 3~11 行代码中的函数在前面已经讲过，`split` 函数将数据框切片分组。第 4 行代码是一个含有两个参数的自编函数 `afterdegreefun`，函数的主体是一个 `if-else`

语句。第 6 行代码判断如果数据框 `x` 包含两个及以上的情感词，那么通过 `substr` 函数提取每一个“切分片段”的前 4 个字符。然后使用 `regexpr` 函数查询任意一个第一类程度副词，如果查到则返回词汇所在的位置，如果查不到则返回 -1；然后使用 `ifelse` 函数判断，如果 `k` 大于 0，则说明找到了第一类程度副词，赋值为 2，如果没找到则赋值为 0，然后筛除第一个搜索结果。因为第一类程度副词总是在情感词的后面，第一个片段无论是否找到情感词都应该去除（它是第一个情感词之前的片段），但是这样的话，片段数量就和情感词的个数对不上了（上一步我们已经舍弃了一个片段）。所以在向量末尾添加一个 0 来占位。如果文本情感词的个数少于两个，则直接给 `scoreafter` 赋值为 0，函数的最后一句返回查询结果 `scoreafter`。

第 14 行代码将 `temp`（`fragment` 分组切片）的元素批量传给查询函数 `afterdegreefun`，同时提供第二个参数 `afterdegree`，然后将结果 `unlist`，现在 `scoreafter` 与 `fragment` 等长。

经过以上步骤，基本上处理好第一类程度副词，其实更常见的是第二类程度副词，即在情感词的前面出现的程度副词。在查询第一类程度副词时，我们是在情感词后面的文本片段的前 4 个字符中查询的；在查询第二类程度副词时，需要使用不同的策略。首先需要在情感词之前的片段中查询，但是不再限制在 4 个字符内，如果在情感词之前的片段中不含有分句符号（如句号、逗号、感叹号等），则将在整个片段的分词结果中匹配；如果片段含有分句符号，则首先将片段按照分句符号切分。如“刘强，对刘强的帮助”片段将被分为“刘强”和“对刘强的帮助”，然后筛出最后一个片段，并选取该片段的最后几个字符进行分词，最后在分词结果中匹配第二类程度副词，即将“对刘强的帮助”分词，然后再匹配第二类程度副词。

### • 再次切分

```
1 sentence <- strsplit(fragment$msg, "[[:punct:]]") # 匹配其中任意一个字符：!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~.
2 temp <- lapply(sentence, length)
3 sentence <- mapply(function(x, sentence) sentence[x], temp, sentence)
4 sentence <- lapply(sentence, function(x) ifelse(length(x) > 0, x, "a"))
5 fragment$msg <- unlist(sentence)
```

在上面的代码中，第 1 行代码按照标点符号再次切分句子，这里仅仅匹配英文字符，而中文字符比较复杂，可以不再一一枚举。另外，我们在进行查找的同时，限制了查找区间（1~8 字符，参看下面的代码），可以尝试按照所有的标点符号切分。因为第二类程度副词通常在情感词的前面，按符号切分后只需要在最后一个片段查询即可，所以第 2~3 行代码提取了切分结果的最后一个片段，也就是 `list` 中每个元素的最后一个值。`mapply` 函数表示将参数元素一一对应地应用到前面的匿名函数中，即 `temp` 和 `sentence` 的每个元素一一配对后，作为参数传递给前面的匿名函数。

检查一下数据后会发现，`sentence` 中出现了 `character(0)`，如果不处理直接 `unlist`，即将 `list` 结构的数据变为非 `list` 结构的数据，则这类数据会被移除。所以在 `unlist` 之前，需要将这类数据替换为缺失值来占位，这里就牵涉了 R 里的 `NA`、`NULL` 等对象。`NA` 表示这个数据值缺失了，但可以查看它的数据类型（见如下代码），而 `NULL` 表示既不存在数据类型，也不存在值，表示该数据点没有设置任何内容。`character(0)` 表示数据类型为字符型，而且没有缺失，只是长度为 0 而已。所以，筛选这类数据时使用长度判断是最好的办法。第 4 行代码使用 `lapply` 函数将 `sentence` 中的每一个元素应用

到后面的匿名函数中，匿名函数的主体是一个 ifelse 函数，用于判断参数 x 的长度，如果长度大于零，则保留原参数 x，否则全部替换为一个字符“a”（随意设定的，但不要替换为 NA，NA 在分词时会报错）。第 5 行代码将结果 unlist，然后赋值给数据框 fragment 的 msg 列。

- NULL 和 NA 和 integer(0)和 character(0)

```
1 m <- NA
2 class(m) # 存在数据类型，不存在值
3 is.na(m)
4 m <- as.character(m)
5 class(m)
6 m <- NULL
7 class(m) #不存在数据类型、不存在值
8 m <- as.character(m)
9 class(m)
10 m <- NULL
11 m <- character(0)
12 class(m) #存在数据类型，不等于 NA，长度为 0
13 is.na(m)
14 length(m)
15 m <- integer(0)
16 class(m)
17 is.na(m)
```

因为第二类程度副词在情感词的前面，所以需要在情感词之前的片段中进行匹配。为了防止程度副词和情感词距离太远，比如不在一个分句中，这里将情感词之前的片段按照标点符号进行切分。但这可能还不够，距离情感词很远的程度副词可能对情感词失去了强调修饰作用，所以，按照以往的经验，需要将范围限制在片段末尾的 1~8 个字符。可以统计程度副词和情感词的距离分布，从中选定一个合适的字符串长度。

- 匹配第二类程度副词

```
1 beforedegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感
分析/dict/自备词典/beforedegree.csv", header = T, sep = ",", stringsAsFactors =
F)
2 library(stringr)
3 sentence <- str_extract(fragment$msg, "\\w{1,8}$")
4 sentence[is.na(sentence)] <- "a"
5 pieceterm <- segmentCN(sentence)
6 scorebefore <- lapply(pieceterm, function(x) {
7   w <- x[grepl("\\S", x)]
8   if (length(w) > 0) {
9     m <- x[x %in% beforedegree$msg]
10    ifelse(length(m) > 0, 2, 0)
11   } else 0
12 })
```

```
13 scorebefore <- unlist(scorebefore)
14 table(scorebefore)
```

在上面的代码中，第 1 行代码用于读取第二类程度副词词典。第 3 行代码截取切片片段的最后 1~8 个字符。有些片段是空字符，这样截取之后会产生缺失值，在分词时会报错，所以第 4 行代码将缺失值换成无意义的字符“a”。第 5 行代码对截取的片段分词，分词的结果是一个 list，每一个片段的分词结果是 list 中的一个元素。第 6 行代码开始将分词结果应用于一个匿名函数中，分词时会产生一些特殊的空白符，匿名函数的第一行筛选这些空白符，grepl 函数根据正则表达式返回逻辑值，如果匹配符合则返回真，如果不符合则返回假。“\\S”表示匹配任何一个非空白符，将筛选后的结果赋值给 w。接着是一个 if-else 判断语句，如果 w 的长度大于 0（即非空），则在分词结果中匹配第二类程度副词，并将匹配结果赋值给 m。第 10 行代码使用 ifelse 函数来判断，如果 m 长度大于 0，则说明在分词结果中找到了第二类程度副词，赋值为 2，否则赋值为 0；如果 w 长度不大于 0，则说明 w 中不包含任何词汇，所以直接赋值为 0；最后将查询结果 unlist。发现在 6 万多个片段中找到了将近 5000 个第二类程度副词，成效卓然。

- 计算程度副词得分

```
1 scoredegree <- ifelse(scoreafter + scorebefore > 1, 2, 1)
2 fragment <- cbind(fragment, scoredegree)
```

在上面的代码中，第 1 行代码根据之前建立的规则，视程度副词不具有叠加效果，在一个情感词前后即便找到 100 个程度副词，仍被看作一个来处理。所以，如果第一类程度副词加第二类程度副词之和大于 1，则给该词汇添加程度副词权重 2，否则权重为 1。第 2 行代码将结果 scoredegree 和原来的情感词数据框按列捆绑在一起。其实，程度副词所起的作用有削弱，也有加强，这里统一添加权重为 2 显然有点儿武断（不负责任）。除此之外，还有很多复杂的情况，比如削弱和加强同时出现等。这些暂时不考虑，有兴趣的读者可以再深入探索。

- 匹配否定词

```
1 notdict <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/自备词典/notdict.csv", header = T, sep = ",", stringsAsFactors = F)
2 scorenot <- lapply(pieceterm, function(x) {
3   w <- x[grepl("\\S", x)]
4   if (length(w) > 0) {
5     n <- x[x %in% notdict$term]
6     ifelse(length(n)%2 == 0, 1, -1)
7   } else 1
8 })
9 scorenot <- unlist(scorenot)
```

查询匹配否定词的原则和查询第二类程度副词的原则几乎一样，直接使用上一步的分词结果 pieceterm 即可。在上述代码中，第 1 行代码读取否定词典。第 2 行代码将分词结果应用到匿名函数。匿名函数的主体部分和查询第二类程度副词的匿名函数相似，先筛选空白符，紧接着是一个 if-else 语句，判断 w 长度是否大于 1，如果是，则筛选出 x 内的否定词，赋值给 n。接下来的 ifelse 函数内略有不同，首先判断找到的否定词的个数，如果个数能被 2 整除，则说明是双重否定，权重赋值为 1，

如果不能被 2 整除，则说明是单纯的否定，权重赋值为-1，如果 x 内没有分出词汇，则权重赋值为 1。

通过以上代码，已经完成了程度副词和否定词的查询了，剩下的任务就是按规则调整。根据之前建立的规则，如果某个情感词的前后出现了程度副词，则将情感词正、负权重中较大者的权重增加一倍，即该词的 PDF 指标和 NDF 指标中较大者的权重增加一倍。

- 程度副词权重调整

```
1 inputdf <- subset(fragment, select = c(id, term, label, scoredegree))
2 inputdf <- join(inputdf, tvpn)
3 temp <- inputdf$pdf > inputdf$ndf
4 inputdf$pdf[temp] <- inputdf$scoredegree[temp] * inputdf$pdf[temp]
5 inputdf$ndf[!temp] <- inputdf$scoredegree[!temp] * inputdf$ndf[!temp]
```

在上面的代码中，第 1 行代码使用 subset 函数取子集。取子集的方法有很多，但是这里要强调的是，取变量时一定要用变量名称取，不要使用变量编号取。第 2 行代码使用 join 函数给情感词匹配上权重。第 3 行代码生成正向权重大于负向权重的布尔序列。第 4 行代码筛选出 PDF 指标较大的词汇，并将其 PDF 指标乘以程度副词的权重。第 5 行代码筛选出 NDF 指标较大或 PDF 指标与 NDF 指标相等的词汇，并将其 NDF 指标乘以程度副词的权重，对于 PDF 指标与 NDF 指标相等的词汇，我们有可能将 NDF 指标权重扩大了一倍，所以，这个分析引擎偏向于将文本归属到情感负向类别内。

下面需要根据否定词调整规则，即制定否定词逆转规则。

- 否定词逆转规则

如果在情感词前面不远处出现了否定词（非双重否定），那么将该情感词的正、负权重进行一次对调，例如“我不喜欢喝茶”，“喜欢”这个词的正向情感权重为 0.05，负向情感权重为 0.02，它的前方出现了“不”，所以“喜欢”的正、负向权重就变成了 0.02 和 0.05。

另外，还会出现否定词+程度副词+情感词的结构，这里先按照程度副词规则调整程度副词的权重，然后根据否定词规则对调正、负权重。当然，有时候还会出现程度副词+否定词+情感词的结构，这里一律武断地视为前一种结构，尽管它们的作用和效果多少有一些不同。有兴趣的读者可以先统计一下两种结构出现的频次，然后再制定调整方式。

在数据挖掘中最忌讳先入为主的偏见看法，比如未进行统计就想当然地认为某种模式更加普遍，要本着“一切规律和知识从数据中来”的原则。

- 否定词逆转调整

```
1 name <- names(inputdf)
2 temp <- inputdf[scorenot == -1, c("id", "term", "label", "scoredegree",
  "ndf", "type", "pdf")]
3 names(temp) <- name
4 inputdf <- inputdf[scorenot == 1, ]
5 inputdf <- rbind(inputdf, temp)
6 inputdf <- subset(inputdf, select = c(id, term, pdf, ndf))
```

在上面的代码中，第 1 行代码提取了列名备用。第 2 行代码首先判断 scorenot 等于-1 所对应的行，因为在这些行的情感词前面发现了否定词，所以提取组的顺序为先提取 NDF，而 PDF 放在了最

后一列，也就是原来的 NDF 指标所在的位置，这样就完成了调换。第 3 行代码更改一下名称，这样原来的 PDF 指标就变成了 NDF 指标，而且列名保持不变。第 4 行代码筛选出 scorenot 等于 1 所对应的行，这些行不需要调整。第 5 行代码直接使用 rbind 函数将它们重新组合在一起，最后使用 subset 函数取子集，这样输入模型的数据基本调整完成了。

其实到现在，使用简单连乘式朴素贝叶斯模型已经基本可以进行测试了，但是如果换成其他模型（需要先对训练数据集做以上处理），比如随机森林、决策树、SVM 等，则会碰到一个问题：比如一条评论里出现了多次“喜欢”，如“我喜欢这家店铺的环境优雅，也喜欢他们的计价方式，但是不喜欢他们的服务”，这里连续出现了 3 次“喜欢”，但是在上述提到的模型中不可能每次出现“喜欢”时就将其作为一个输入变量，需要有一种方法或规则将重复出现的数值转化为一个统一的变量，所以我们设计了连乘规则。

- 连乘规则

如果一条文本中某个情感词连续出现了  $n$  次，那么该情感词的正向权重连乘  $n$  次作为新正向权重，负向权重连乘  $n$  次作为新负向权重。例如“喜欢”这个词的正向情感权重为 0.05，负向情感权重为 0.02，它在一段文本中出现了 3 次，所以“喜欢”的正负向权重就变成了  $0.05^3$  和  $0.02^3$ 。

- 连乘调整

```
1 tempdf <- split(inputdf, inputdf$id)
2 prodfun <- function(x) {
3   temp <- x[duplicated(x$term), "term"]
4   if (length(temp) > 0) {
5     data1 <- x[-which(x$term %in% temp), ]
6     data2 <- x[which(x$term %in% temp), ]
7     data2 <- aggregate(cbind(pdf, ndf) ~ id + term, data = data2, FUN = prod)
8     x <- rbind(data1, data2)
9   }
10  return(x)
11 }
12 inputdf <- lapply(tempdf, prodfun)
13 inputdf <- do.call("rbind", inputdf)
14 inputdf <- inputdf[complete.cases(inputdf), ]
```

在上面的代码中，第 1 行代码将数据框 inputdf 按照 id 列分组形成一个 list，list 中的每一个元素对应一条文本的分词结果数据框，然后将 list 的每一个元素作为 x 参数应用到自编函数 prodfun 里。从第 2 行代码开始是执行连乘的自编函数，函数的第一行筛选文本分词中重复的词汇并赋值给临时变量 temp。接下来是一个 if-else 句式，先利用 temp 的长度判断文本中有无重复词汇，如果长度大于零，则说明存在重复词汇，然后剔除重复词汇并将剩余的行赋值给 data1 备用，同时筛出重复的词汇赋值给 data2。接着对 data2 中的 pdf、ndf 列按 id 和 term 分组后应用于函数 prod，求累乘积。然后将 data1 和 data2 重新合并为原来的数据框。如果 temp 长度不大于零，则说明 x 中不存在重复的词语，不做任何改变，最后返回 x 即可。第 12 行代码使用 lapply 函数将 tempdf 中的每一个元素应用于自编函数。第 13 行代码使用 do.call 函数，调用 rbind 函数将新产生的 list 转为数据框，并重新赋



值给 inputdf。第 14 行代码用于筛除 inputdf 中含有缺失值的行，要记得有一些模型是对缺失值敏感的。到这里规则调整已经基本完成了，如果使用其他算法，则同样要对数据的训练数据集进行规则调整，然后输入模型训练，而且需要保证训练数据集和测试数据集的变量一致。

其实这是一个非常浪费时间的环节，上面的 prodfun 函数完全被带入了一个误区（修改自一个同学的代码），虽然耗时耗资源，但是方便理解，在后面会讲解它的替代思路。

### 10.5.3 朴素贝叶斯情感分析器

下面建立一个简单的朴素贝叶斯分类器，其规则如下：将一段文本中所有情感词汇的正向权重的乘积作为该文本的正倾向值，所有情感词汇的负向权重的乘积作为该文本的负倾向值，如果文本的正倾向值大于负倾向值，则归为情感正向，反之则归为情感负向（包括相等的情况）。如果你觉得文本情感倾向整体上符合某一概率分布，比如 7 : 3，当然也可以乘以这类先验概率，然后再进行正负倾向的比较。

- 朴素贝叶斯分类器

```
1 result <- aggregate(cbind(pdf, ndf) ~ id, data = inputdf, FUN = prod)
2 result$Bayes <- ifelse(result$pdf > result$ndf, 1, -1)
3 result <- join(result, unique(fragment[, c("id", "label")]))
4 table(result$label, result$Bayes)
#      -1      1
# -1 1805  156
#  1   315 1625
5 result <- transform(result, negscofill = ndf/(pdf + ndf))
```

在上述代码中，第 1 行代码使用透视表 aggregate 函数完成正、负向权重连乘的任务，首先将 pdf、ndf 按照 id 分组，然后使用 prod 分别求累乘积。第 2 行代码使用 ifelse 函数比较文本的正、负倾向值的大小，如果正倾向值大，则赋值为 1，如果负倾向值大，则赋值为 -1；如果文本的正、负倾向整体属于不均匀分布，比如正向文本和负向文本的比例为 7 : 3，则可以分别将正倾向值乘以 0.7，负倾向值乘以 0.3 之类的先验概率，然后再进行比较。第 3 行代码按照 id 给结果关联人工标记的情感倾向。第 4 行代码使用 table 函数分析朴素贝叶斯分类和人工标注的差异。从结果上看，人工共标注了 1961 个负向文本，机器标注正确的有 1805 个，占比为 92%；人工标注了 1940 个正向文本，机器标注正确的有 1625 个，占比为 83.7%，总体正确率为 88%，效果还是不错的。第 5 行代码将正负倾向值百分比化，可以将其看作给一个文本所打的情感分（负向），当然这个情感分是否和文本的情感强度相关还有待检验。

以上是基于加权词典规则的半监督式情感分析引擎，通过上面的分析，可以看出其中还有不少可以提高的地方，当然不符合统计逻辑的地方也不少。如果读者觉得还可以改进，则可以尝试对上述分析过程中出现的遗漏进行有的放矢的改进。

## 10.5.4 支持向量机（SVM）、决策树等情感分析器

以上简单构建了一个朴素贝叶斯模型，但是如果使用其他算法，则可能还有一些细节需要调整。下面就构建一个支持向量机（SVM）模型，以方便读者了解其中的调整细节，如果下面的路径能够走通，则同样可以移植到其他算法，例如随机森林、决策树（C4.5 算法和 CART 算法）等。

### 1. SVM 情感分析模型训练

与之前的朴素贝叶斯模型不同，这次需要先准备训练数据集。

- 训练数据集分词整理

```
1 tvpn <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/
自备词典/tvpn.csv", header = T, sep = ",", stringsAsFactors = F)
2 train <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/train.csv", sep = ",", header = T, stringsAsFactors = F)
3 sentence <- as.vector(train$msg)
4 sentence <- gsub("[[:digit:]]*", "", sentence) #清除数字[a-zA-Z]
5 sentence <- gsub("[a-zA-Z]", "", sentence)
6 sentence <- gsub("\\.+", "", sentence)
7 train <- train[!is.na(sentence), ]
8 sentence <- sentence[!is.na(sentence)]
9 train <- train[!nchar(sentence) < 2, ]
10 sentence <- sentence[!nchar(sentence) < 2]
11 library(Rwordseg)
12 insertWords(dict)# 沿用上一节 dict 对象
13 system.time(x <- segmentCN(strwords = sentence))
14 temp <- lapply(x, length)
15 temp <- unlist(temp)
16 id <- rep(train[, "id"], temp)
17 label <- rep(train[, "label"], temp)
18 term <- unlist(x)
19 trainterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)
20 trainterm <- trainterm[trainterm$term %in% tvpn$term, ]
21 library(plyr)
22 trainterm <- join(trainterm, train[, c("id", "msg")])
```

- 训练集句子切分

```
1 train <- split(trainterm, trainterm$id)
2 splitsentence <- function(x) {
3   term <- x$term
4   nnum <- nchar(term)
5   term <- term[order(nnum, decreasing = TRUE)]
6   if (length(term) > 930) {
7     msg <- x$msg[1]
8     for (i in 1:length(term)) {
```

```

9      msg <- sub(term[i], 't', msg)
10     }
11     fragment <- strsplit(msg, "t")
12     x$msg <- unlist(fragment)[1:length(x[,1])]
13   } else {
14     term <- paste(unique(term), collapse = "|")
15     fragment <- strsplit(x$msg[1], term)
16     x$msg <- unlist(fragment)[1:length(x[,1])]
17   }
18   return(x)
19 }
20 temp <- lapply(train, splitsentence)
21 #fragment <- do.call("rbind", temp)
22 library(data.table)
23 fragment <- rbindlist(temp)
24 fragment <- as.data.frame(fragment, stringsAsFactors = F)
25 fragment[is.na(fragment$msg), "msg"] <- "a"

```

在上面的代码中，仅自编切分函数有一处不同，即 `splitsentence` 函数内部增加了一个判断，如果一个文本中的情感词超过 930 个使用循环切分函数，则可以先将所有的情感词替换为 `t`，然后碰到 `t` 就切分一次；如果不超过 930 个情感词使用循环切分函数，则仍然按照原来的方法切分。

另外，当 `list` 增大时，使用 `rbind` 函数将它们粘贴在一起效率很低，特别是当 `list` 中包含因子变量时，它将比对因子水平，从而造成工作效率直线下降。所以这里使用了 `data.table` 包中的 `rbindlist` 函数替换了 `rbind` 函数。

- 查找第一类程度副词

```

1 afterdegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分
析/dict/自备词典/afterdegree.csv", header = T, sep = ",", stringsAsFactors = F)
2 afterdegree <- paste(afterdegree$msg, collapse = "|")
3 temp <- split(fragment, fragment$id)
4 afterdegreefun <- function(x, afterdegree) {
5   if (length(x$term) > 1) {
6     piece <- substr(x$msg, 1, 4)
7     k <- regexpr(afterdegree, piece)
8     scoreafter <- ifelse(k > 0, 2, 0)
9     scoreafter <- scoreafter[-1]
10    scoreafter <- c(scoreafter, 0)
11  } else scoreafter <- 0
12  return(scoreafter)
13 }
14 temp <- lapply(temp, afterdegreefun, afterdegree)
15 scoreafter <- unlist(temp)

```

- 再次切分

```

1 sentence <- strsplit(fragment$msg, "[[:punct:]]") # 匹配其中任意一个字

```

符: !"#%&'()\*+,-./:;<=>?@[\\]^\_`{|}~.

```
2 temp <- lapply(sentence, length)
3 sentence <- mapply(function(x, sentence) sentence[x], temp, sentence)
4 sentence <- lapply(sentence, function(x) ifelse(length(x) > 0, x, "a"))
5 fragment$msg <- unlist(sentence)
```

- 匹配第二类程度副词

```
1 beforedegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/自备词典/beforedegree.csv", header = T, sep = ",", stringsAsFactors = F)
```

```
2 library(stringr)
3 sentence <- str_extract(fragment$msg, "\\w{1,8}$")
4 sentence[is.na(sentence)] <- "a"
5 pieceterm <- segmentCN(sentence)
6 scorebefore <- lapply(pieceterm, function(x) {
7   w <- x[grepl("\\S", x)]
8   if (length(w) > 0) {
9     m <- x[x %in% beforedegree$msg]
10    ifelse(length(m) > 0, 2, 0)
11   } else 0
12 })
13 scorebefore <- unlist(scorebefore)
14 table(scorebefore)
```

- 计算程度副词得分

```
1 scoredegree <- ifelse(scoreafter + scorebefore > 1, 2, 1)
2 fragment <- cbind(fragment, scoredegree)
```

- 匹配否定词

```
1 notdict <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/自备词典/notdict.csv", header = T, sep = ",", stringsAsFactors = F)
```

```
2 scorenot <- lapply(pieceterm, function(x) {
3   w <- x[grepl("\\S", x)]
4   if (length(w) > 0) {
5     n <- x[x %in% notdict$term]
6     ifelse(length(n)%2 == 0, 1, -1)
7   } else 1
8 })
9 scorenot <- unlist(scorenot)
```

- 程度副词权重调整

```
1 inputdf <- subset(fragment, select = c(id, term, label, scoredegree))
2 inputdf <- join(inputdf, tvpn)
3 temp <- inputdf$pdf > inputdf$ndf
4 inputdf$pdf[temp] <- inputdf$scoredegree[temp] * inputdf$pdf[temp]
5 inputdf$ndf[!temp] <- inputdf$scoredegree[!temp] * inputdf$ndf[!temp]
```

- 否定词逆转调整

```
1 name <- names(inputdf)
2 temp <- inputdf[scorenot == -1, c("id", "term", "label", "scoredegree",
  "ndf", "type", "pdf")]
3 names(temp) <- name
4 inputdf <- inputdf[scorenot == 1, ]
5 inputdf <- rbind(inputdf, temp)
6 inputdf <- subset(inputdf, select = c(id, term, pdf, ndf))
```

- 乘调整

```
1 temp <- data.table(inputdf)
2 system.time(temp <- temp[,lapply(.SD, prod),by=c("id", "term"), .SDcols
= c("pdf", "ndf")])
3 # system.time(inputtrain <- aggregate(cbind(pdf, ndf) ~ id + term, data
= inputdf, FUN = prod))
4 inputtrain <- as.data.frame(temp, stringsAsFactors = F)
5 inputtrain <- inputtrain[complete.cases(inputtrain), ]
```

这里对前面的代码进行了很大的调整，完全替换了之前使用 `prodfun` 函数的思路。其实没有必要判断一个词是不是重复出现，只需要将 `pdf` 指标和 `ndf` 指标按照 `id` 和 `term` 分组连乘透视即可，从而减少了很多周折。这个案例告诉我们：如果某个环节运行速度特别慢，则首先应该考虑是不是思路错了，是否违反了语言的特征。在上面的代码中，第 1 行代码将数据框转为 `data.table` 对象。第 2 行代码中的 `.SD` 表示为原 `data.table` 按照分组列分出的小组（`subset`），这些 `subset` 不包含用于分组的列（`id`、`term`），然后使用 `lapply` 函数将这些小组应用于 `prod` 累乘函数。其中 `by` 用于指定分组列，`.SDcols` 用于指定被分组列，其效果和下面注释掉的 `aggregate` 是一样的，速度却提高了很多倍。第 4 行代码将结果转化为数据框。第 5 行代码剔除含有缺失值的行。

下面需要对数据进行一次调整，我们称之为变量分裂，即一个情感词分裂为两个输入变量，也就是说情感词有一个正向值和一个负向值。为了保证它们作为衡量指标都能输入模型，需要将一个词分裂为两个变量。比如对于“喜欢”，输入“喜欢”的正向值时，作为变量“p 喜欢”；输入“喜欢”的负向值时，作为变量“n 喜欢”。

- 变量分裂

```
1 library(data.table)
2 ptrain <- dcast(data = as.data.table(inputtrain[, c("id", "term", "pdf")]),
formula = id ~ term, fun.aggregate = sum, value.var = "pdf")
3 temp <- paste("p", names(ptrain), sep = "")
4 names(ptrain) <- c("id", temp[-1])
5 ntrain <- dcast(as.data.table(inputtrain[, c("id", "term", "ndf")]), id
~ term, sum)
6 temp <- paste("n", names(ntrain), sep = "")
7 names(ntrain) <- c("id", temp[-1])
8 ptrain <- as.data.table(ptrain)
9 ntrain <- as.data.table(ntrain)
```

```

10 temp <- as.data.table(unique(fragment[, c("id", "label")]))
11 setkey(ptrain, id)
12 setkey(ntrain, id)
13 setkey(temp, id)
14 pntrain <- merge(ptrain, ntrain)
15 pntrain <- merge(pntrain, temp)
16 pntrain <- as.data.frame(pntrain)
17 pntrain$label <- as.factor(pntrain$label)
18 pntrain <- subset(pntrain, select = -id)

```

在上面的代码中，第 2 行代码将数据由 long 型转换 wide 型。dcast 函数来自 data.table 包，速度比 reshape2 包快一点，但是要求数据必须是 data.table 对象，所以 data 参数在赋值时使用了 as.data.table 函数进行了转化。其他参数和 reshape2 包一样，formula 指定转换的列；id ~ term 即保持 id 列不变；term 由行转变为列，每一个 term 变为一个变量；value.var 用于指定填充单元格的变量，即透视变量 pdf；fun.aggregate 用于指定对透视变量做什么样的操作，即调用哪个函数，这里对透视变量求和，这样对正向权重的转换就完成了。但是一个词需要变成两个变量，所以需要在词的前面加一个字符“p”，表示其代表正向值变量，所以第 3 行代码代码给 ptrain 的每一列的名称前添加了一个字符“p”，生成一个备用的列名称向量 temp。但 ptrain 的第 1 列是 id 列，不需要更改其名称，否则后面无法关联，所以 4 行代码去掉 temp 的第 1 个元素 pid，然后用“id”字符补上，再命名给 ptrain 即可。第 5~7 行代码对情感词负向值做相同的操作，唯一不同的是在列名称前添加一个字符“n”作为区分。为了快速进行关联（join）操作，第 8~10 行代码将相应的对象转化为 data.table 对象。因为设置了主键的关联匹配速度更快，所以第 11~13 行代码使用 setkey 函数将 ptrain、ntrain、temp 中的 id 列设为主键。第 14~15 行代码对数据进行了关联合并，merge 函数在未设置 all 参数的情况下默认取交集。合并完成之后，第 16 行代码将 data.table 转化为数据框，其实不进行转化，data.table 对象也具有数据框的属性，但是无奈很多模型不能识别。第 17 行代码将 label 列的数据类型改为因子，因为如果因变量的数据类型为因子，SVM 模型就将其当作分类任务处理，如果为数值变量，就会将其当作回归任务处理。第 18 行代码删除 id 列，否则其会作为自变量输入模型。

需要提醒的是，dcast 函数在只有 4GB 内存的电脑上已经无法完成任务了，需要迁移到服务器上完成，这里提供一个笨方法：将原来的数据框分批转换为宽表，但是每一批的数据转换为宽表后，变量需要保持一致。

### • SVM 建模

```

1 library(e1071)
2 set.seed(35)
3 y <- pntrain$label
4 x <- subset(pntrain, select = -label)
5 system.time(svmmodel <- svm(x, y, kernel="radial", cost = 1, gamma = 0.01))
#      user      system elapsed
# 91480.653    95.245 91726.092
#tuned <- tune.svm(x, y, kernel="radial", gamma = 10^(-7:-1), cost = 10^(1:3))

```

在上面的代码中，第 2 行代码使用 `set.seed` 设定随机种子。第 3 行代码将标签设为因变量。第 4 行代码将其他数据设为自变量。第 5 行代码使用 `svm` 函数建模，外面嵌套了 `system.time` 函数用于搜集建模所用时间；`kernel` 用于设置核函数；`gamma` 用于设置松弛变量，调节分类间隔，其参数越大表示容忍错误的程度越大；`cost` 为惩罚因子，其参数越大表示越不容许出错（我认为 `cost` 为单个值，表示对所有分错点的惩罚是一样的，但是在工作中我们可能更看重某一类型的点，对错误的容忍度更低。那么就要求 `cost` 是一个权重向量，但是我暂时没发现函数能这么设置）。整个建模过程需要在服务器上进行（在单机上很难处理完成这么大数据量以及这么多维度），共耗时 25 个小时。最后一行注释掉的代码用于寻找最优参数（`gamma` 和 `cost`），找到错误最低时的松弛变量 `gamma` 和惩罚因子 `cost`。这是一个内置了交叉检验的过程，不用担心过拟合，而是应该担心电脑的性能和消耗的时间。

- 训练集效果

```
1 prediction <- predict(svmmodel, x)
2 table(prediction, y)
#           y
# prediction  -1     1
#           -1 11787    54
#           1   74 11704
```

我们看到 SVM 模型对训练数据集的拟合准确率很高，着实让人眼前一亮。能否准确地描述历史数据不能成为判断模型是否优秀的标准，模型好不好得看模型对未来的描述，或者是对未来的预测能力，所以我们要用模型预测一下未来。用前面构建的 SVM 模型预测一下测试数据集，首先要将测试数据集整理一下。

## 2. SVM 情感分析模型测试

历尽千辛万苦总算可以小试牛刀了。

- 测试数据集分词整理

```
1 tvpn <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析/dict/
自备词典/tvpn.csv", header = T, sep = ",", stringsAsFactors = F)
2 test <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/test.csv", sep = ",", header = T, stringsAsFactors = F)
3 sentence <- as.vector(test$msg)
4 sentence <- gsub("[[:digit:]]*", "", sentence) #清除数字[a-zA-Z]
5 sentence <- gsub("[a-zA-Z]", "", sentence)
6 sentence <- gsub("\\\\.*", "", sentence)
7 test <- test[!is.na(sentence), ]
8 sentence <- sentence[!is.na(sentence)]
9 test <- test[!nchar(sentence) < 2, ]
10 sentence <- sentence[!nchar(sentence) < 2]
11 library(Rwordseg)
12 insertWords(dict)# 沿用上一节 dict 对象
13 system.time(x <- segmentCN(strwords = sentence))
14 temp <- lapply(x, length)
```

```

15 temp <- unlist(temp)
16 id <- rep(test[, "id"], temp)
17 label <- rep(test[, "label"], temp)
18 term <- unlist(x)
19 testterm <- as.data.frame(cbind(id, term, label), stringsAsFactors = F)
20 testterm <- testterm[testterm$term %in% tvpn$term, ]
21 library(plyr)
22 testterm <- join(testterm, test[, c("id", "msg")])
23 test <- split(testterm, testterm$id)
24 splitsentence <- function(x) {
25   term <- x$term
26   nnum <- nchar(term)
27   term <- term[order(nnum, decreasing = TRUE)]
28   if (length(term) > 930) {
29     msg <- x$msg[1]
30     for (i in 1:length(term)) {
31       msg <- sub(term[i], 't', msg)
32     }
33     fragment <- strsplit(msg, "t")
34     x$msg <- unlist(fragment)[1:length(x[,1])]
35   } else {
36     term <- paste(unique(term), collapse = "|")
37     fragment <- strsplit(x$msg[1], term)
38     x$msg <- unlist(fragment)[1:length(x[,1])]
39   }
40   return(x)
41 }
42 temp <- lapply(test, splitsentence)
43 library(data.table)
44 fragment <- rbindlist(temp)
45 fragment <- as.data.frame(fragment, stringsAsFactors = F)
46 fragment[is.na(fragment$msg), "msg"] <- "a"
47 afterdegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感
分析/dict/自备词典/afterdegree.csv", header = T, sep = ",", stringsAsFactors = F)
48 afterdegree <- paste(afterdegree$msg, collapse = "|")
49 temp <- split(fragment, fragment$id)
50 afterdegreefun <- function(x, afterdegree) {
51   if (length(x$term) > 1) {
52     piece <- substr(x$msg, 1, 4)
53     k <- regexpr(afterdegree, piece)
54     scoreafter <- ifelse(k > 0, 2, 0)
55     scoreafter <- scoreafter[-1]
56     scoreafter <- c(scoreafter, 0)
57   } else scoreafter <- 0

```



```

58   return(scoreafter)
59 }
60 temp <- lapply(temp, afterdegreefun, afterdegree)
61 scoreafter <- unlist(temp)
62 sentence <- strsplit(fragment$msg, "[[:punct:]]") #匹配其中任意一个字
符: !"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~.
63 temp <- lapply(sentence, length)
64 sentence <- mapply(function(x, sentence) sentence[x], temp, sentence)
65 sentence <- lapply(sentence, function(x) ifelse(length(x) > 0, x, "a"))
66 fragment$msg <- unlist(sentence)
67 beforedegree <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感
分析/dict/自备词典/beforedegree.csv", header = T, sep = ",", stringsAsFactors =
F)
68 library(stringr)
69 sentence <- str_extract(fragment$msg, "\\w{1,8}$")
70 sentence[is.na(sentence)] <- "a"
71 pieceterm <- segmentCN(sentence)
72 scorebefore <- lapply(pieceterm, function(x) {
73   w <- x[grepl("\\S", x)]
74   if (length(w) > 0) {
75     m <- x[x %in% beforedegree$msg]
76     ifelse(length(m) > 0, 2, 0)
77   } else 0
78 })
79 scorebefore <- unlist(scorebefore)
80 scoredegree <- ifelse(scoreafter + scorebefore > 1, 2, 1)
81 fragment <- cbind(fragment, scoredegree)
82 notdict <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/dict/自备词典/notdict.csv", header = T, sep = ",", stringsAsFactors = F)
83 scorenot <- lapply(pieceterm, function(x) {
84   w <- x[grepl("\\S", x)]
85   if (length(w) > 0) {
86     n <- x[x %in% notdict$term]
87     ifelse(length(n)%2 == 0, 1, -1)
88   } else 1
89 })
90 scorenot <- unlist(scorenot)
91 inputdf <- subset(fragment, select = c(id, term, label, scoredegree))
92 inputdf <- join(inputdf, tvpn)
93 temp <- inputdf$pdf > inputdf$ndf
94 inputdf$pdf[temp] <- inputdf$scoredegree[temp] * inputdf$pdf[temp]
95 inputdf$ndf[!temp] <- inputdf$scoredegree[!temp] * inputdf$ndf[!temp]
96 name <- names(inputdf)
97 temp <- inputdf[scorenot == -1, c("id", "term", "label", "scoredegree",

```

```
"ndf", "type", "pdf"]])
  98 names(temp) <- name
  99 inputdf <- inputdf[scorenot == 1, ]
 100 inputdf <- rbind(inputdf, temp)
 101 inputdf <- subset(inputdf, select = c(id, term, pdf, ndf))
 102 temp <- data.table(inputdf)
 103 system.time(temp <- temp[,lapply(.SD, prod),by=c("id", "term"), .SDcols
= c("pdf", "ndf")])
 104 inputtest<- as.data.frame(temp, stringsAsFactors = F)
 105 inputtest <- inputtest[complete.cases(inputtest), ]
```

以上整理测试数据集的流程和整理训练数据集的流程没有任何区别，这里不做过多解释。但是在整理测试数据集时有一点需要特殊说明，即测试数据集变量要与训练集变量保持相同，比如测试数据集可能因为数据量比较少，缺少了某些情感词，或者出现了一些稀有情感词，这时就需要对测试数据集进行增减情感词了，即将缺少的补上，将多余的删除，否则其将不能正确地输入模型。

### • 变量持同

```
1 addterm <- unique(inputtrain$term)
2 inputtest <- inputtest[inputtest$term %in% addterm,]
3 temp <- unique(inputtest$term)
4 addterm <- addterm[!addterm %in% temp]
5 n <- length(addterm)
6 temp <- rep(0, n*length(inputtest))
7 temp <- data.frame(matrix(temp, nrow = n))
8 temp[, 2] <- addterm
9 temp[, 1] <- "add"
10 names(temp) <- names(inputtest)
11 inputtest <- rbind(inputtest, temp)
```

在上面的代码中，第 1 行代码提取训练数据集中所有的变量并去重。第 2 行代码使用 %in% 函数在 addterm 中查找 inputtest\$term，如果查到就将相应的数据提取出来，重新赋值 inputtest，这样就筛选除了那些仅在测试数据集出现的情感词。第 3 行代码提取测试数据集的情感词汇，并去重。第 4 行代码筛出只在训练数据集中出现的词汇，重新赋值给 addterm。第 5~10 行代码制造一个文本，id 编号“add”（第 9 行代码），并生成了一个  $n \times 4$  的数据框。第 8 行代码将 addterm 赋值给数据框 temp 的第 2 列，然后使用 inputtest 各列的名称命名数据框 temp 的列。第 11 行代码将两个数据框捆绑在一起，这样测试数据集的变量就和训练数据集完全一致了，经过调整之后我们只需要将 id 为“add”行删除即可。

### • 变量分裂

```
1 library(data.table)
2 ptest <- dcast(data = as.data.table(inputtest[, c("id", "term", "pdf")]),
formula = id ~ term, fun.aggregate = sum, value.var = "pdf")
3 temp <- paste("p", names(ptest), sep = "")
4 names(ptest) <- c("id", temp[-1])
```

```

5 ntest <- dcast(as.data.table(inputtest[, c("id", "term", "ndf")] ), id ~
term, sum)
6 temp <- paste("n", names(ntest), sep = "")
7 names(ntest) <- c("id", temp[-1])
8 ptest <- as.data.table(ptest)
9 ntest <- as.data.table(ntest)
10 temp <- as.data.table(unique(fragment[, c("id", "label")]))
11 setkey(ptest, id)
12 setkey(ntest, id)
13 setkey(temp, id)
14 pntest <- merge(ptest, ntest)
15 pntest <- merge(pntest, temp)###取交集，移除了 id 为 add 的行，因为 temp 中不
包含这个 id
16 dim(pntest)
17 pntest <- as.data.frame(pntest)
18 pntest$label <- as.factor(pntest$label)

```

这一步的代码和训练数据集变量分裂时的代码一样，不再赘述，只是其中 merge 取交集时删除了我们添加的辅助数据“add”。现在可以准备测试模型了。

- 测试结果

```

1 temp <- subset(pntest, select = -c(id, label))
2 prediction <- predict(svmmodel, temp)
3 label <- pntest$label
4 table(prediction, label)
#           label
# prediction  -1    1
#           -1  372   62
#            1 1589 1878
5 rate <- sum(diag(table(prediction, label)))/sum(table(prediction, label))
#0.5767752

```

在上面的代码中，第 1 行代码使用 subset 函数剔除 id、label 等无关列。第 2 行代码使用 predict 函数预测测试数据集结果。第 3 行代码使用 table 函数制作混淆矩阵。让人大跌眼镜，擅长处理文本数据的 SVM 出现了过拟合，负向情感的文本预测结果太差劲了。第 5 行代码使用 diag 函数取混淆矩阵对角线上的数值，并使用 sum 函数求和。结果就是预测正确的样本数，对混淆矩阵求和就是总样本数，二者相除即为准确率。让人大跌眼镜的是准确率不到 60%，这就是突破重重艰险训练了 25 小时，速度奇慢的 SVM。

### 10.5.5 如何选择支持 SVM 的核函数

既然这里用到了 SVM，那么显然无法回避如何选择 SVM 的核函数这个问题。其实我们的生活中不乏要做出艰难决定的时刻，令人尴尬的是，刻意逃避也是其中一个可能造成严重后果的选项。

首先，核函数用来计算两个向量在隐式映射过后的空间中的内积，可以简化映射空间中的内积运算。SVM 的一个特征就是将数据映射到高维空间，但是在高维空间计算向量积时，计算量会随纬度的增加而增加，这显然是一件麻烦的事儿，核函数正好可以解决这个问题。

一般情况下，常见的 SVM 的核函数分为以下几类：

- (1) 线性核函数 (Linear)
- (2) 多项式核函数 (Polynomial)
- (3) 高斯核函数 (Gaussian: RBF)
- (4) Sigmoid 函数

除此之外还有很多其他核函数，这么多的核函数，该如何选择？其实这类问题很常见，例如如何选择随机森林的树数？如何选择主题模型的主题数？如何防止决策树过拟合？……诸如此类的问题归根结底还是模型优选、模型评价、模型监控的问题。一般情况下，可以分为以下 4 个步骤进行评估。

(1) 评估数据类型是否符合备选模型的要求，比如有些模型要求变量是连续的，如果不符合，则选择符合数据类型要求的模型即可，如果符合，则进行下一步。

(2) 构建一个评价指标：比如准确率、复杂度、NMSE、平均绝对方差、Geni 系数等。

(3) 使用交叉检验对任意一个确定的备选项构建模型并计算评价指标。

(4) 对评价指标进行统计检验，选出评价指标最优的模型。

按照上面的步骤回答问题，虽然不一定得满分，但至少让面试官觉得你熟悉解决问题的流程，拥有解决问题的思维方式。

当数据为非线性可分时，毫无疑问，RBF 核函数 (Radial Basis Function) 是首选，因为 SVM 的本质是一个线性分类器，只不过当数据为线性不可分时，其将数据映射为一个有限的高维空间，然后寻找一个可分的超平面。但是确定核函数最好的方法还是按照上面的交叉检验流程，因为没有比这个方法更加优秀的筛选体系了。

下面通过 SVM 构建一个酒品分类器，顺便介绍一下挑选核函数的分析流程。其实在前面已经使用相似的方法分析过随机森林的树数。

- k 层交叉检验数据集

```
1 library(plyr)
2 CVgroup <- function(k, datasize, seed) {
3   cvlist <- list()
4   set.seed(seed)
5   n <- rep(1:k, ceiling(datasize/k))[1:datasize]
6   temp <- sample(n, datasize)
7   x <- 1:k
8   dataseq <- 1:datasize
9   cvlist <- lapply(x, function(x) dataseq[temp==x])
10  return(cvlist)
11 }
```

```

12 wine <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 10 章情感分析
/data/wine.csv", header = T)
13 k <- 10
14 datasize <- nrow(wine)
15 cvlist <- CVgroup(k = k, datasize = datasize, seed = 1206)

```

上面的代码已经在前面分析随机森林的过程中讲过了，其主要用于生成交叉检验的数据集，这里不再赘述。需要说明的是，原始数据被分成了 10 组，也就是进行 10 层交叉检验。

- 层交叉检验

```

1 data <- wine
2 library(plyr)
3 j <- c("linear", "polynomial", "radial", "sigmoid")
4 i <- 1:k
5 i <- rep(i, times = length(j))
6 j <- rep(j, each = k)
7 x <- data.frame(i, j)
8 cvtest <- function(i, j) {
9   library(e1071)
10  train <- data[-cvlist[[i]],]
11  test <- data[cvlist[[i]],]
12  train$cultivars <- as.factor(train$cultivars)
13  model <- svm(cultivars ~ ., data = train, kernel = j)
14  prediction <- predict(model, subset(test, select = - cultivars))
15  cultivars <- test$cultivars
16  rate <- sum(diag(table(cultivars,prediction)))/sum(table(cultivars,
prediction))
17  temp <- data.frame(rate)
18 }
19 system.time(pred <- mdply(x, cvtest))

```

在上面的代码中会用到 plyr 包中的 mdply 函数，这个函数是 apply 函数的扩展，当需要传递多个参数时，就可能用到了它。mdply 函数可以将一个数据框按行传递给需要调用的函数，作为该函数的参数，也就是说，这个数据框的每一行一一对应到多参数函数的相应参数。其中第 1 行代码将数据赋值给 data。第 3 行代码生成核函数向量。第 4 行代码生成数据集编号。第 5 行代码将数据集编号重复 length(j) 次，即有几个核函数，数据集编号就重复多少次。第 6 行代码将核函数向量重复 k 次，也就是有多少个数据集，核函数就重复多少次。这里需要注意 rep 函数的参数 times 和 each 对结果的影响。第 7 行代码将它们捆绑为数据框，每一行即可作为相应参数传递给多参数自编函数 cvtest。cvtest 是一个半成品函数，之所以说它是半成品函数，是因为它会在环境中寻找名为 data 的数据集，这显然不具有可移植的特征。如果要将其用于其他数据集，则需要对函数进行修改，所以它是一个半成品函数（不建议写这样的函数）。cvtest 函数包括两个参数，分别对应 x 的两列数据。第 9 行代码调取包含 SVM 的包 e1071。第 10 行代码提取训练数据。第 11 行代码提取测试数据。第 12 行代码将酒品类型列转化为因子，因为只有因变量的数据类型为因子时，SVM 才会将之作为分类任务，

否则会作为回归任务。第 13 行代码构建 SVM 模型。第 14 行使用模型预测测试数据集，predict 的第 1 个参数为已经构建好的模型，第 2 个参数是和训练数据集结构相同的测试数据集，当然也可以预测训练数据集，subset 函数在前面出现过多次了。第 15 行代码提取测试数据集的酒品类型列备用。第 16 行代码使用 table 函数将实际酒品类型和预测结果做混淆矩阵，使用 diag 函数取混淆矩阵对角线上的数值，并使用 sum 函数求和，结果就是预测正确的样本数，对混淆矩阵求和就是总样本数，二者相除即为准确率。第 17 行代码将准确率赋值给 temp 数据框。第 19 行代码使用 mdply 函数将数据框 x 按行传递给函数 cvtest，并作为参数，然后调取并执行函数。其外面嵌套了一个抓取系统时间的函数 system.time，用于计算整个过程消耗的时间，通常在测试时使用。

有的读者在看函数帮助文档时感觉无从下手。其实了解某个函数的用法当然要看它的参数和自己的数据的对应关系，看它要求的数据类型是 list 还是数据框，抑或是向量。所有参数是不是来自于同一个数据集。如果不是，应该怎么调整。如果是，它们和自己的数据之间的对应关系是什么。搞懂了这些，基本上才算了解这个函数的用法。

### • 方差分析

```
1 pred$j <- as.factor(pred$j)
2 bartlett.test(rate ~ j, data = pred)
# Bartlett test of homogeneity of variances
#
# data: rate by j
# Bartlett's K-squared = 3.9151, df = 3, p-value = 0.2708
3 temp <- aov(rate ~ j, data = pred)
4 summary(temp)
#           Df Sum Sq Mean Sq F value Pr(>F)
# j           3 0.00567 0.001889   1.479  0.237
# Residuals   36 0.04598 0.001277
5 TukeyHSD(temp)
6 library(dplyr)
7 pred %>% group_by(j) %>% summarise(mean(rate))
# Source: local data frame [4 x 2]
#           j mean(rate)
# 1 linear 0.9601307
# 2 polynomial 0.9604575
# 3 radial 0.9774510
# 4 sigmoid 0.9882353
```

在上面的代码中，第 1 行代码首先将分组变量转化为因子。第 2 行代码使用 bartlett 方法检验准确率的方差齐性， $F$  值越小， $P$  值越大。 $P$  值大于 0.05，就证明没有差异，说明方差齐。接着使用 aov 函数对准确率进行方差分析。接着使用 summary 函数分析，显示差异不显著，说明不同核函数的 SVM 模型在这个数据集上预测准确率差异不显著（ $P$  值远远大于 0.05），即没有必要做多重检验了。但是为了展示整个分析流程，这里使用 TukeyHSD 函数进行多重比较，各组之间的  $P$  值都远远大于 0.05，因此各组之间不存在差异，这说明在这个数据集上，使用任何核函数的预测效果差异都

不显著。

尽管从准确率均值上看，线性核函数比较弱，但是它们在统计上的差异并不显著，当你的数据变量较多并且本身就是线性可分的，那么线性核函数是不错的选择。使用高斯核函数构建 SVM 模型效果同样好，但是我会选择前者。因为，第一，前者是参数模型，而后者不是。第二，高斯核函数构建的 SVM 模型复杂程度和训练数据集的大小相关，因此训练高斯核函数的压力就更大，而且需要保证核矩阵时刻可以调用。第三，最让人提心吊胆的是，复杂模型往往更容易过拟合。另外，线性核函数对于处理稀疏矩阵，特别是文本类型的数据集，效果比较好（可以尝试使用线性核函数构筑情感分类器，但是我可以负责任地告诉你，效果也不好）。

如果数据集不是线性可分的，那么使用线性分类器就比较强人所难了，显然这时应该使用高斯核函数了。所以从理论上说，线性核函数用于处理线性数据集，非线性数据集就交给高斯核函数吧。那么问题来了，什么样的数据集才是线性可分的呢？简单地说，即能被一次函数分开的数据集就是线性可分的。那么怎么判断一个数据集是否线性可分的？答案是构建一个线性分类器。所以，很多理论判断最终还是要落实到实践中，而上述内容就是对实践过程的简单描述了。

当然关于支持向量机，除了筛选核函数，还要设定最优的惩罚因子和松弛向量，上面已经讲过 `tune.svm` 函数，但是如果数据量大，那么使用这个函数估计要等十天或半个月才能得出结果。

### 10.5.6 情感分类器方法评价

在本节的结尾不得不评价一下各种算法了，虽然这会得罪算法和其背后的使用者，但是作为整个数据挖掘流程中不可避免的一部分，也不能避而不谈。下面所用的训练数据集和测试数据集都是前面用过的，而且朴素贝叶斯模型是在具有 4GB 及以上内存的笔记本电脑中完成的，其他模型均是在服务器上完成的。

#### 1. 随机森林模型

前面使用训练数据集构建了随机森林模型，其中包含 100 棵树，每棵树的选择变量为 73 个，在训练数据集中的预测正确率为 83.31%，在测试数据集中的预测正确率为 85.31%，建模时间大概需要 30 分钟，预测时间可以忽略不计。

#### 2. SVM 模型

10.5.5 节重点介绍的基于高斯核函数的 SVM 模型建模花费了 25 小时，出现了过拟合现象，其在训练数据集中的预测正确率达到 99.4%，而在测试数据集中的预测正确率仅达到了 57%（每次建模波动较大），但在最好的情况下预测正确率也不足 60%。

#### 3. 朴素贝叶斯模型

而使用简单的朴素贝叶斯模型，建模时间基本可以忽略，在测试数据集中的预测正确率却高达 88%，速度也有了惊人的改变。另外，无论是预测正确性还是计算速度，此模型都有很多可以优化

的地方，比如使用分布式并行计算等。

### 4. 决策树\_C4.5 模型

而构建决策树\_C4.5 模型，在训练数据集中的预测正确率为 85.69%，在测试数据集中的预测正确率为 81.24%。

从预测正确率来看，随机森林在训练数据集和测试数据集中的预测效果都比较好，但模型构建时间较长。决策树 C4.5 模型在训练数据集中的正确率最高，在测试数据集中的正确率比随机森林模型差一些，但是运行速度明显高于随机森林模型。朴素贝叶斯模型的总体预测正确率仅次于随机森林模型，建模时间要比随机森林模型少。SVM 模型的效果最差。

除了上述关键点，朴素贝叶斯模型的优越性在于不需要占用大量内存的过程，即没有短板效应。另外，它输出的算是一个评分制，这样我们可以尝试将结果和情感的强度关联。所以，这里比较推荐朴素贝叶斯模型。需要提醒读者的是，模型的准确性和变量的权重相关，以上我们只是使用了 DF 作为权重，感兴趣的读者可以使用卡方作为权重进行建模预测，也许会有另一种结果（据说有人做出来的结果还不错，尽管还是没有超越上述方法）。

## 10.6 谈谈情感分析的下一步思考

经过前面的介绍，我们已经熟悉了情感分析的常见方法，并通过自己的思考组建了一个比较高效的情感分析引擎。但这并非意味着学习就此结束了，还有很多有待解决的问题。还可以思考以下几个方面的内容。

### 1. 让准确率再上一个台阶

情感分析的准确率还有可以改善的空间。首先，前面使用的训练数据集还是比较小，我们可以收集更多的训练数据集（不限行业）。其次，我们还可以寻找合适的替代 DF 权重的评估方式。这样做的目的是保证每一个情感词的权重更加真实，这样加权情感词库才会更加符合人们日常用语的习惯，从而提高准确率。

另外，可以寻找新的情感词库，扩大情感词库的覆盖范围。同时需要设计自动发现情感词的算法，比如情感词应该具有这样的特征：在正向文本中出现的概率与在负向文本中出现的概率差异较大时，可以尝试根据这个假设发现情感词的算法。

还可以设计更加细腻的规则体系，比如前面使用的程度副词并没有分门别类地添加权重，但事实上它们是有区别的，比如“非常”和“有点儿”表达的情绪明显不一样，为其添加权重可以使分析结果更加细致准确。



### 2. 更加细腻的情感分析方式

要进行更加细腻的情感分析可能需要从整个分析流程上进行调整，首先待选算法输出的结果必须是连续的，因为分类型算法能够识别的类别总是有限的（需要人工定义多类别训练数据集），显然其细腻程度比不上使用连续的指标衡量情感程度，但是这就需要算法输出的连续变量能够真正衡量一个文本的情感程度。当然，如果要使用这类算法，衡量情感词权重的方式可能也需要进行调整，不幸的是，我还没发现这样的算法，甚至连设计思路都没有。

### 3. 情感归属问题

在一段话中可能有多个主体词，比如有人在一条评论中夸奖了 A，鄙视了 B，那么怎样将正向情感归属于 A，将负向情感归属于 B 呢？最容易想到的思路是通过切割的方式碎片化文本，然后通过语法分析获得句子的主题词。但是，在现实中存在大量缺少主体词的文本，比如，文本中并没有提到××公司，但是人们一眼就能认出是在说哪家公司。对于这类文本可以使用话题模型。首先对文本进行话题分类，在理想情况下，讨论 A 的文本自然被归到 A 的话题中，鄙视 B 的文本自然被归到 B 的话题中，文本一旦被分类了，文本所附带的情感自然也有了归属，至于 LDA 话题模型会在第 11 章中重点介绍。

以上是对情感分析下一步方向的一些思考。

---

注：情感分析的整体应用的代码我们已经进行了优化，请参见本书的资源下载文件。

---

# 第 11 章

## 话题模型： 很多牛人过不去的坎儿

怎样快速学习一门编程语言？作为一名数据挖掘工程师，我也经常思考这个问题，可能这个问题的答案因人而异，有些人看看书马上就能写程序，有些人看了很多教程，参加了无数次培训，仍然不能独立地写出一个脚本。根据我的经验，学习编程语言比较快速的方式就是在实际应用中学习。

之前在学习 R 语言时，我花费了大量时间阅读书籍、教程和观看视频，但收效甚微。在清洗数据过程中，一个非常小的问题就能击破我的看似无懈可击的知识体系，最终，我的知识技能还是在实际的应用项目中巩固起来的。

后来学习 Python 等其他语言时，我不再一本本地看书了，而是看一下入门教程就开始设定一个项目去完成，缺少什么再去学习什么。我发现这样的学习方式对我来说是最快速的，所以推荐大家不妨试试。

现实中的数据分析不像书中的那些示例，在实际中，你会遇到各种垃圾数据。因此，我认为只有通过实践才能获得长足的成长。

### 11.1 话题模型与文案文本集

好了，废话不多说，从本章起我们开始进入另一个实战案例，这次给自己设定的项目是从网络的一个文案文本集中探究该文集的内容。

首先要定义一下这个问题，这个问题实质上可以看成是一个文本挖掘的小项目，问题可以完整地表述为怎样从这个网络方案文本集中挖掘出来知识点？那么我们不妨把这个问题分解一下：

(1) 数据收集：解决这个问题的基础是获得这些网络文案文本集，这里不使用自动抓取的方式，而是手动搜集了一些文本数据。

(2) 我们要准备一些周边数据，比如有多少份文本集，共涉及多少个参与者等。这个过程除了

对解决问题有一定的支撑作用，也能帮助我们探索数据，了解数据。一份报告除了需要靓丽的核心解决方案，周边的背景知识也能起到锦上添花的效果。

(3) 首先要知道这些文本集讨论了哪些话题。这个过程肯定不能遍历文本进行总结，而且话题也无法准确定义，可以使用话题模型解决这个问题，其实这是一个发现话题并将文本归类的问题。

(4) 文本集讨论的话题是不是随着时间发生了变化？这个问题可以根据文本的词汇，尝试将不同时期的文本聚类，如果聚类有很明显的时间特征，则说明在不同的年代文本讨论的内容是不同的。

(5) 既然有阶段性变化，那么各个阶段都讨论了什么？这个问题可以根据上一步得出特殊阶段的文本，画出各个阶段的话题词云。

通过以上 5 步，基本上可以就这个问题给出一个相对有条理的答案。而本章就是以解决这些问题为目的，让读者学习分析解决问题的思路，同时学习一些提高 R 语言分析效率的窍门。

### 11.1.1 任务仍然是以处理 dirty data 开始

首先，需要将文本数据载入，共有 47 份文本，分别对应于 47 个 txt 文件，存放在同一个文件夹内。我们需要完成的任务有 3 个：(1) 获取文件路径；(2) 按名称批量读入数据；(3) 将文件名称作为 id 和文本内容组成数据框。

- 获取文本路径

```
1 reviewpath <- "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型是很多牛人过不去的坎/data/文案文本集"
2 completepath <- list.files(reviewpath, pattern = "*.txt$", full.names = TRUE)
```

在上面的代码中获取文本路径的函数已经在前面详细讲解了，这里就不赘述了。

- 按名次批量读入数据

```
1 read.txt <- function(x) {
2   des <- readLines(x)
3   n <- grep("各位代表", des, perl = T)#每次正文都是以各位代表开头
4   des <- des[seq(n[1], length(des), 1)]#提取相关内容
5   return(paste(des, collapse = ""))
6 }
7 review <- lapply(completepath, read.txt)
8 docname <- list.files(reviewpath, pattern = "*.txt$")
9 reviewdf <- as.data.frame(cbind(docname, unlist(review)),
10                           stringsAsFactors = F)
11 colnames(reviewdf) <- c("id", "msg")
```

在上面的代码中，需要注意的是，readLines 函数读取文本文件是分行读取的，我们平常所说的“一段文字”在 txt 文件中只是一行，也就是说，des 是一个包含这个文档所有段落的 list，可以将函数解体仅读取一个文件，然后索引这个 list，查看每一段具体的文字。比如，要索引 des 的第一段文字，则代码为 des[1]。R 的索引是从 1 开始编码的。

`grep` 函数是本章要介绍的第一个正则表达式函数。正则表达式函数就是将一系列的规则当成模式，然后匹配数据的内容，其目的是找到符合一定模式的数据内容。比如，在这里我们发现每一份文本的正文内容都是以“各位代表”这 4 个字开始的，所以，我们需要找到每篇文章中“各位代表”这 4 个字出现的位置，然后将其第一次出现的位置（行编号）之后的内容作为本次文本分析的正文部分。`grep` 函数返回所有“各位代表”出现的行编号，并赋值给了 `n`，这里使用了 `grep` 的 3 个参数，第 1 个参数是要查找的模式，第 2 个参数是要遍历的内容，第 3 个参数是否指定 Perl 语言兼容模式。如果设置 `perl` 参数兼容为真，则表示要查找的模式在 Perl 语言中也是可以正常执行的，大家尽量写 Perl 语言兼容的正则表达式并设置 `perl` 参数为 `Ture`，因为当数据达到千万级别时，`perl` 参数兼容模式的速度优势会非常明显。

`seq` 函数用于产生等差序列，其第 1 个参数是序列的起点，第 2 个参数是序列的终点，第 3 个参数用来设置步长。`seq(n[1], length(des), 1)` 的意思就是以向量 `n` 的第 1 个数为起点，以 `des` 的长度（`des` 这个向量有多少个元素，也就是这篇文章有多少段，包括空白段）为终点，产生一个步长为 1 的序列，然后使用这个序列作为编号（索引），将 `des` 中所有对应编号的元素（文本中指段落）提取出来，再赋值给 `des`，原来 `des` 中的内容就被替换了。

`paste` 函数将 `des` 中的内容粘贴在一起，也就是说，粘贴后 `list` 中的各元素成为一个整体，不能通过索引查找，而且它们之间不设置任何分隔符。最后用 `return` 函数将结果返回。在自编函数过程中产生了其他结果，但没有返回，因此无法在内存中查到，当然你也可以返回多个结果，例如 `return(des, n)` 中就返还了两个值，返还的结果是一个 `list` 包含着 `des` 和 `n`。

使用 `lapply` 函数将完整路径一个个地传递给我们自建的函数 `read.txt`，后者按路径一个个读取并将结果并作为元素赋值给 `review`。`review` 是一个 `list`，可以按索引查看，例如 `review[1]`。不要直接查看 `review` 的所有内容，因为这 47 份文本数据还是比较大的，计算机配置低的话会死机。

`review` 是由多个文本组成的一个 `list`，每一个元素就是一份文案。`list.files` 函数用于读取文件名，并和文件内容（`unlist` 解散为 `vector`）合并捆绑成数据框。第 10 行代码对数据框重命名。

在 R 中，向量是通过 `c` 函数建立的，其小括号内如果是数字或已经存在的对象，就不用引号，如果是字符，就要用引号。`data.frame` 的行编号（即行名称）是不能重复的，因为 `data.frame` 靠它完成索引，另外这里将文章的名称作为 `id`。

### 11.1.2 数据清洗

完成了数据读入，还需要对数据进行简单的清洗。数据清洗要和数据挖掘的目的相匹配，比如在前面将整篇文章的段落合并后就不能看出文本段落之间的变化，如删除一些英文字符，就有可能失去了某些英文缩写提供的特殊信息等。我有一个习惯，在项目开始之前先画项目的技术路线，详细到每一个细节，在画的过程中原本不怎么清晰的思路就逐渐清晰化了，这样代码就按照技术路线走。如果有某些内容要改正，就看一下技术路线，从而可以从全局上把握在哪个环节修改。

- 数据清洗

```
1 reviewdf$msg <- gsub(" ", "", reviewdf$msg)
2 reviewdf$msg <- gsub("(鼓掌)", "", reviewdf$msg)
```

上面的代码用到了以正则表达式为基础的筛选和替换函数。这里用到了 `gsub` 函数，其作用是将符合模式的文本替换为指定的内容，其中第 1 个参数指明正则表达式的模式（被替换的内容），这里仅仅为了替换空格，只需要用 “ ” 作为参数就可以了。第 2 个参数指定替换后的内容，用引号括起来，这里替换为空，即直接输入 “ ” 就行了。第 3 个参数就是被操作的对象。综合起来的意思就是将 `reviewdf$msg` 中的空格替换为空，然后再赋值给 `reviewdf$msg`，记住，这时 `reviewdf` 数据框里的 `msg` 列已经被改变。

第 2 行代码和第 1 行代码结构一样，将数据 `reviewdf` 的 `msg` 列中的 “(鼓掌)” 替换为空，因为不是文本的内容，而是对听众反应的一种描述，所以要把它替换为空，以免影响结果。

- 数据清洗

```
1 sentence <- reviewdf$msg
2 id <- reviewdf$id
3 sentence <- gsub("[[:digit:]]*", "", sentence)
4 sentence <- gsub("[a-zA-Z]", "", sentence)
5 id <- id[!is.na(sentence)]
6 sentence <- sentence[!is.na(sentence)]
7 msg <- as.data.frame(cbind(id, sentence), stringsAsFactors = F)
```

在上面的代码中，第 1、2 行代码将 `msg`、`id` 列提取出来，然后仍然用 `gsub` 函数替换一些模式。`[[:digit:]]*` 是指替换所有的英文字母及数字（这些需要考虑一下是否真的需要这么做）。替换所有英文的正则表达式为 `[a-zA-Z]`，上一步已经替换英文了，之所以重复做。是想告诉读者替换英文的正则表达式还可以这么写。然后要判断 `sentence` 是不是一些元素完成以上的替换后就只剩下空值，处理网络评论的文本时，可能会存在整条评论都是英文的情况，被替换后就成了空值。因为，我们的生活中毕竟存在大量满口中英混杂的“混音人士”，在空值分词的时候会报错，所以要把 `sentence` 中为空的值删除。`is.na` 函数返回一个和对象一样长的布林值序列，如果为空，就为 `True`，如果为非空，就为 `False`。那么我们提取一个向量时就按真或非来提取，但是要保留的是非空值，所以，这里在 `!is.na(sentence)` 之前加了一个 “!”，表示非的函数。这个符号是将 `Ture` 变为 `False`，反之亦然。当然你也可以用布林值提取、筛选数据框。向量值的另外一种选择方式和数据框比较相似，比如，要筛选向量的前 3 个元素，即 `sentence[1:3]`，其和数据框的不同点就是少了逗号分隔。顺便提一下，`list` 的筛选和向量一样，但是如果是嵌套 `list`，即 `list` 的元素是向量或者是另一个 `list`，那么筛选到具体的对象则为 `x[[1]]`。

虽然替换后为空的概率很少，但是如果构建的是一个自动化的分析系统，就需要防止这种错误中止程序，因为 `sentence` 的每一个元素都对应于一个 `id`，所以，如果一旦 `sentence` 的某个元素为空被删除，那么也应该把它对应的 `id` 删除。因此，在移除 `sentence` 的空值之前就应该用 `!is.na(sentence)` 把 `sentence` 为空的 `id` 先移除，再移除 `sentence` 的空值（想一想为什么），然后再把 `id` 和 `sentence` 按列 `cbind` 在一起，将其转化为数据框（`as.data.frame`）。

如果想熟悉一下正则表达式，则推荐去一个网站学习一下基础知识（在百度中搜索“三十分钟让你学会正则表达式”）即可找到。

另外，我们需要读取停用词表，停用词就是对研究作用不大的词，在此过程中需要把它去除，以节省计算空间，如下代码所示。

```
1 stopwords <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：
很多牛人过不去的坎儿/data/stopword.csv", header = T, sep = ",")
```

## 11.2 话题模型中几个重要的数据处理步骤

数据经过简单的清洗之后就可以进行数据挖掘的前期处理了。首先要对中文分词，之后需要对数据表进行一次长表、宽表的变换，然后利用 `tm` 包将分词结果转化为文本词频库，再通过 `tfidf` 算法对词汇进行一次筛选，在这个过程中会穿插介绍一些有关 LDA 算法。

### 11.2.1 中文分词

分词应该算是文本挖掘最基础的一步了，一般使用 `Rwordseg` 包进行分词。我们的任务有以下几个：首先对文本分词并将分词结果和文本 `id`（名称）关联起来形成数据框；然后去除停用词、单字；最后生成词频矩阵。

比较有名的中文分词包非 `Rwordseg` 和 `jieba` 莫属，它们采用的算法大同小异，这里选择 `Rwordseg` 包。

```
1 library(Rwordseg)
2 segment.options(isNameRecognition = TRUE)#设置人名识别
3 system.time(x <- lapply(sentence, function(x) segmentCN(x, nature =
TRUE)))
4 msgWords <- as.data.frame(cbind(rep(msg[,1], unlist(lapply(x, length))),
unlist(x),
names(unlist(x))), stringsAsFactors = F)
5 rownames(msgWords) <- seq(1:length(msgWords[,1]))
6 names(msgWords) <- c("Docid", "Term", "Nature")
```

在上面的代码中，`segment.options` 函数用来设置是否对人名进行识别，将其设置为真，表示文本中如果有人名就按照人名的规则分词。在文本中提到的人名、地名均具有一定的时代特征，在某些年代的文本中出现的可能性比较大，所以这里将 `segment.options` 设置为真。

`lapply` 函数就不再讲解了，它的作用是将 `sentence` 的元素一个个地传递给分词函数，这里使用了一个匿名函数。由于匿名函数的主体部分只有一句“`segmentCN(x, nature = TRUE)`”，所以没用大括号。这句代码的意思很简单，就是用 `lapply` 函数将 `sentence` 一个个地作为 `x` 传递给匿名函数，然后函数的主体分词函数 `segmentCN` 分离 `x` 中的词汇。注意，这里设置了 `nature = TRUE` 参数，要求在

分词的同时识别词的词性。要了解标注词性的英文字母所代表的词性，可以查看 Rwordseg 包的帮助文档。

这里需要看一下分词的结果到底是什么。下面这句代码表示分别对两个文本进行分词。执行结果 `m` 是一个嵌套的 list，一个文本的分词全部包含在 list 的一个向量内，向量包括分词和词性。分词包作者将词性设置为词的名称，因此可以很方便地提取词性。例如，`names(m[[1]])`就提取了第一个文本所有词汇的词性。当然你也可以将 list 解散，然后提取全部词性。

```
1 m <- segmentCN(c("我真的不懂是怎么分词的", "分词后的结果是什么样的呢"), nature = TRUE)
```

分词结束之后要将每一篇文章的词语和文章 id 关联起来，形成一个 data.frame。前面说过 lapply 函数是将 sentence 的元素一个个地传递出去，那么分词结果 list 的第一个向量就对应 sentence 的第一个元素，也就是第一篇文章，我们只需要知道第一个向量包含多少个字，假设为 `n`，然后将文章 id 重复 `n` 次，就可以把它们按列绑定为一个 data.frame。

```
1 msgWords <- as.data.frame(cbind(rep(msg[,1], unlist(lapply(x, length))),
                                unlist(x),
                                names(unlist(x))), stringsAsFactors = F)
```

上面这句代码比较长，这里把它分解一下。首先我们要知道分词结果中每个向量的长度，然后把对应的文章 id 重复相应的次数就可以了。`lapply(x, length)`就是获得 `x` 里每个向量（一个向量对应一篇文章的分词结果）的长度，返回的结果是一个 list。我们要先把它分解为向量，即使用 `unlist` 函数。`msg` 的第一列是对应的文章 id，我们将其提取出来复制对应的次数，就形成了一个和分词结果等长的向量。比如，第一篇文章分出了 10000 个词，我们将第一篇文章的 id 复制 10000 次，依此类推，就可以和词绑定为 data.frame 了。使用 `rep(msg[,1], unlist(lapply(x, length)))`就完成了复制对应文章 id 的任务。`rep` 函数用于复制一个序列，比如 `rep(1, 2)`就是将 1 重复两次，`rep(c(1,2), c(2, 4))`就是将 1 重复两次，然后将 2 重复 4 次。将分词结果整个解散（`unlist(x)`）就可以和刚生成的文章 id 序列绑定在一起了。通过 `unlist(lapply(x, length))`，也可以查看每篇文本中所包含的词数。`unlist(x)`为解散 `x` 成一个向量，然后作为词汇列。`names(unlist(x))`为提取所有分词对应的词性，再加上前面千辛万苦生成的文本 id 列，然后用 `cbind` 函数将它们绑定在一起。最后使用 `as.data.frame` 函数将之转化为 data.frame。

下面用 `rownames` 重新给每一行编号，一句话有多少行就编多少号。这里使用的是 `seq` 函数。奇怪的是，我刚开始学习 R 的时候老是分不清 `seq` 和 `rep` 这两个函数，虽然知道它们的英文是什么意思。

最后一行代码给数据框 `msgWords` 的各列命名，修改列名的函数包括 `names` 或 `colnames`。

#### • 停用词处理

```
1 msgWords <- msgWords[!(msgWords$Term %in% stopwords$term),]#去除停用词
2 msgWords <- msgWords[which(nchar(msgWords$Term) > 1),]
3 msgWords$Term <- gsub("[\r\n]", "", msgWords$Term)
```

首先要去除停用词，停用词在 11.1 节已经载入内存了，存放在 `stopwords` 这个数据框里，其中 `term` 列就是我们收集到的停用词。如果 `msgWords$Term` 在 `stopwords$term` 里能找到，就要把这个词

从 msgWords 数据框中去除，也就是删除它所在的行。这里用到 R 里的匹配函数，第一个匹配函数是 %in%，比如 a%in%b 表示从对象 a 中取出元素在 b 中查找，如果找到就返回 TRUE，反之则返回 FALSE，最终返回一个和 a 等长的布林序列。按照我们的需求，只需要保留那些在 stopwords\$term 找不到的词就行了，所以在 !(msgWords\$Term %in% stopwords\$term) 中加了一个 “!”，然后按行筛选就行了。这里需要提醒读者的是，函数 %in% 在数据量比较大时处理速度也会慢下来，建议使用 data.table 包中的 %chin% 函数来替代，处理速度要快一些。

第 2 行代码保留词长大于 1 的词，要做到这一点首先要计算字符的长度，R 里面是用 nchar 完成的，它是 number character 的缩写，这与其他语言不同（包括 Python）。Python 一般使用 len，但 R 里的 length 函数是对元素计数的，which(nchar(msgWords\$Term) > 1) 是用 which 函数筛选字符长度大于 1 的行。which 函数很容易，返回符合判断语句的元素编号，小括号里是一个判断语句；最后一行是最常见的函数了，gsub 函数将一些词后面的换行符替换为空值（不是空格）。

这里分词和前期整理就结束了，下面做一个非常重要的数据整型。

### 11.2.2 数据整型

下面继续我们的文本挖掘任务，第一步就面临着将一个 long 型数据转换为 wide 型数据。

- 文本挖掘整型

```
1 library(reshape2)
2 library(plyr)
3 Logic <- rep(1, length(msgWords[,1]))
4 msgWords <- as.data.frame(cbind(msgWords, Logic), stringsAsFactors = F)
5 msgTerm <- dcast(msgWords, Term ~ Docid, value.var = "Logic", fun.aggregate
= length)
6 msgTerm <- melt(msgTerm, id = 1)
7 msgTerm <- msgTerm[which(msgTerm$value > 0),]
8 names(msgTerm) <- c("Term", "Docid", "Freq")
9 msgTerm <- join(msgTerm, msgWords[,1:3])
10 msgTerm <- unique(msgTerm)
```

先要生成一个词频表，包括 3 列：文本 id、词语和词频，即表示每篇文章中出现某个词汇的次数。这个表可以通过 table 函数生成交叉表然后整理实现。首先将之前的文本——词汇表，由 long 型转换为 wide 型，同时对词汇计数，转换的结果是行为词汇，列为文本 id，每个单元格为文本中出现某词汇的词频（次数）。

再给原来的表加一个辅助列 logic，使用 rep 函数生成一个长度和 msgword 行数相同且所有值均为 1 的数列 logic，然后把它添加到原有的数据框中。第 5 行代码使用 dcast 函数对 msgwords 进行转换，Term 列作为 id 列位置保持不变。拆分列为 Docid 列，数值列为刚刚生成的 Logic 列，这里使用的透视函数是 length 函数，也可以试用一下 sum 函数，效果是一样的。当然现在这种情况可以不需要设置透视函数，因为默认的就是 length 函数，可以使用 head 函数查看一下 msgTerm 数据框被 dcast



转换成了什么样了。从第 6 行代码开始，再将宽表转化为 long 型表。这里使用 melt 函数，参数分别为数据框和 id 列的编号，将第 1 列作为 id 列，其他列全部融合就可以了。融合之后很多词在某篇文章中出现的次数为 0，这类记录就不需要了。第 7 行代码使用 which 语句筛选并保留了 value(词频)大于 0 的记录。

然后对数据框的列重新命名。第 8 行代码依然使用 names 函数对列进行命名，但这里落下了原来数据框中的一列数据(在文本挖掘中比较重要)，即词性列，所以要给分词重新配上词性。第 9 行代码使用 join 函数将两个数据框中的列按照 id 匹配合并在一起。在默认状态下，join 函数会把两个数据框中所有列名称相同的列作为 id 列进行匹配(如果你不指定 id 列)。通过 Term，Docid 作为 id(其实只用 Term 作为 id 即可)将 msgTerm 与 msgWords 的前 3 列合并在一起，这样就获得了词性列 nature。另外，join 函数分为 inner、left、right 和 full 模式，一般默认为 left 模式(左联接)。第 10 行代码对整个文档去重，使用的是 unique 函数，它以数据的所有列为依据去重。

### 11.2.3 怎样设定“阈值”

建立 LDA 模型需要先将 data.frame 格式转换为 DocumentTermMatrix 格式，这里我们将两种数据格式连接在了一起。其优点是可使用数据框对数据进行预处理，包括删除一些停用词。如果直接用 tm 包，那么当我们面对计算机内存不足，不知怎么操作时，只能武断地删除部分低频词，而如将数据框和 tm 包联用，就可以先检查数据再删除不需要的词。

- data.frame 与 DocumentTermMatrix 之间相互转换

```
1 x <- msgWords[, 1:2]
2 <- split(x$Term, x$Docid)#数据框转化为 list
3 wordcorpus <- Corpus(VectorSource(x)) # 组成语料库格式
4 dtm <- DocumentTermMatrix(wordcorpus,
                             control = list(
                               wordLengths=c(2, Inf), # 限制词长
                               bounds = list(global = c(5,Inf)), # 设置词的最小频率
                               removeNumbers = TRUE,
                               weighting = weightTf,
                               encoding = "UTF-8"))
```

将 data.frame 格式转换为 DocumentTermMatrix 格式，首先要经过一次中间转换，即先转换为 list。list 中的每个元素是一个向量，向量对应一个文本的所有分词结果(这里是指我们整理后的分词结果)，然后再将 list 转化为 DocumentTermMatrix 格式。在上面的代码中，第 1 行代码提取词汇列和文本 id 列。第 2 行代码使用 split 函数将数据框转化为 list。其实 split 函数采用的是透视表分组的方法，其中用到两个参数：第 1 个参数指定被分组的对象列(可以是多列，也可以是整个数据框)，第 2 个参数指定分组的编号——id 列，将 id 相同的词汇聚成一个组，作为最终表单 list 的一个元素。

形成 list 后，可以通过 VectorSource 函数和 Corpus 函数转换成语料库了。VectorSource 函数是将 Vector 对象转化为 Source 对象，Corpus 函数再将 Source 对象转化为 Corpus 对象，可以通过 inspect

函数查看 Corpus 对象。arules 包中也有一个 inspect 函数。

这里重点介绍一下 DocumentTermMatrix 函数，它将产生一个压缩了的矩阵，像一个以词和文档关联的词频点袋子。其中第 1 个参数是上一步刚刚形成的 Corpus 对象。control 这个参数里有各种可以调整的参数，作为参数的 list 列表。wordLengths 用来限定词的长度，这里将词长设置为大于或等于两个字符。bounds 里用来设置词频，该词频不仅指的是在全部文档中出现的次数，还指单个文档中出现的次数，这里设置最小词频为 5。removeNumbers 用来设置是否移除数字。encoding 用来设置编码方式，这样第一个 dtm 文本词频矩阵就形成了。

下一句代码很难理解，首先要理解什么是 TFIDF，比如有一篇《中国抗战阅兵时间表》的新闻，下面提取这篇文章的关键词。比较简单有效的方法就是根据词频（Term Frequency，TF）提取，TF 就是一篇文章中出现某个词的次数（当然如果是进行文章之间的横向比较，因为文章有长有短，则还要除以文章的总词数）。你可能认为“中国”的出现次数最多，其实不然，“的”“是”“在”“地”之类的词是出现次数最多的，这类词是停用词，在提取关键词之前需要剔除。剔除停用词之后，“中国”就成了出现词频最多的词，但是看一下题目会发现，“中国”在所有的汉语文章中出现的可能性都很大，而“抗战”“阅兵”“时间表”才是更关键的关键词，所以要给 TF 一个权重。于是“逆文档频率”（Inverse Document Frequency，IDF）横空出世，如果一个词比较“常见”（指在日常所有文档中），那么它的 IDF 就比较低。要计算 IDF，首先要有一个充实的语料库。利用 IDF 作为惩罚权重，就可以计算词的 TFIDF。这样比较常见的“中国”在《中国抗战阅兵时间表》这篇文章中的 TFIDF 就会被拉低，而其他稀有词汇的 TFIDF 就会相对提高，再按照 TFIDF 排序即可以方便地找出关键词了。

这里为什么要计算 TFIDF？目的是利用 TFIDF 删除一些重要性相对较弱的词。

- 利用 tm 包计算 TFIDF

```
1 library(tm)
2 library(slam)
3 term_tfidf <- tapply(dtm$v/row_sums(dtm)[dtm$i], dtm$j, mean) * log2
(nDocs(dtm)/(col_sums(dtm > 0) + 1))
```

每篇文章的每个词都有一个 TFIDF，比如“春风”这个词在 3 篇文章中都出现过，那么就要算出 TFIDF 均值。在上面的代码中，dtm\$v 为每个词在每篇文章中出现的次数，row\_sums(dtm) 为每篇文章的总词数，dtm\$i 为每个词频点的文章编号。注意，这里指的是词频点，不是词，row\_sums(dtm)[dtm\$i] 形成了和 dtm\$v 等长的文章总词数向量序列。

什么是词频点？比如一个词在 3 篇文章中出现的频率分别为 1、2、3，那么它就有 3 个词频点，在原来我们常见的词频文档矩阵里，列为文档编号，行为词语，或者列为词语，行为文档编号，每一个方格（cell）里的数值即为词频。dtm 有 3 个属性，dtm\$v 为每个词在每篇文章中出现的次数序列，dtm\$i 为每个词频点的文章编号，dtm\$j 为每篇文章的每个词的编号序列，它们都是等长的。

其中 dtm\$v/row\_sums(dtm)[dtm\$i] 计算出了每一个词在每一篇文章中的 TF，然后使用 tapply 函数按 dtm\$j 分组，即按词分组求词汇在所有文章中的 TF 均值。如果你仔细研究就会发现，对 TFIDF 求均值其实就是对 TF 求均值，因为只要语料库不变，一个词的 IDF 是不会变的。这里使用 tapply

函数直接对 TF 求均值, `tapply` 有 3 个参数, 即数据、与数据等长的分组向量和计算函数。这里用 `dtm$j` 作为分割 (group) 向量, 就是将每篇文章每个词的 TF 按词求均值。

`nDocs(dtm)` 求出了 `dtm` 中共包含几篇文档, 也就是语料库中文章的总篇数。`col_sums(dtm > 0)` 求出每个词在语料库中出现的文档数, 加 1 是为了防止分母为零, 前后相除之后求对数的结果就是 IDF。

到此, 我们就算出了 TFIDF 均值。

前面说过计算 TFIDF 均值的目的是用来筛选词汇, 但是应该保留 TFIDF 均值为多少的词汇呢? 也就是说指标计算好了, 但是阈值怎么设定呢? 考验一个人的统计功底和活学活用能力的时候到了, 要设定阈值, 首先考虑以下两个问题。

(1) 指标的统计意义, 即看一看指标的分布, 比如人群收入的分布、消费能力的分布, 通过考察指标的分布可以确认具有统计意义的阈值, 比如选择均值还是中位数, 抑或是众数?

(2) 指标的业务意义, 即使选择的阈值再好, 没有业务意义也是不行的。比如, 如果产品的目标人群是高端消费者, 而你非得把阈值设在消费能力中位数以下, 显然是不靠谱的, 这时你要找出为你贡献 80% 消费额的那一小部分人群。

#### • 数据准备

```
1 summary(term_tfidf)
2 ll <- term_tfidf >= quantile(term_tfidf, 0.5)
3 dtm <- dtm[,ll]
4 dtm <- dtm[row_sums(dtm)>0, ]
5 summary(col_sums(dtm))
```

在通过 `summary` 函数返回的统计指标中, 比较均值和中位数可以大概了解 TFIDF 为偏分布, 为了保存尽量多的词汇, 应该选取中位数作为阈值。在上面的代码中, 首先计算 `term_tfidf` 的中位数, 这里使用了 `quantile` 函数, 它是用来计算百分位数的, 第 1 个参数为数据; 第 2 个参数是计算的百分位点, 可以是单个值 (如上面代码所示), 也可以用一个向量一次计算多个百分位点 (如下面代码所示, `seq` 函数产生一个数值为 0.1~1, 步长为 0.1 的等差序列向量)。

```
quantile(term_tfidf, seq(0.1, 1, 0.1))
ll <- term_tfidf >= median(term_tfidf)
```

除了使用 `quantile` 函数计算中位数, 当然也可以使用 `median` 函数。在上面的代码中, 第 1 行代码计算出中位数之后, 以原序列和中位数进行比较 (`>=`), 返回一个和原序列等长的布林序列, 如果原序列的值大于或等于中位数, 则返回真 (True), 反之则返回非 (False)。

第 2 行代码就是使用返回的布林序列从文档词频矩阵 (`dtm`) 中筛选出大于 TFIDF 均值的中位数的词。中位数这个界限并非固定, 如果你的模型没有达到预期的效果, 也许这个中位数可以换成其他统计量帮助你优化模型 (实质上是优化结果), 其实你也可以设计其他指标, 比如 TF。

在优化一个系统之前, 首先要熟悉系统中影响产出的节点。

经过调整 TFIDF 后, 可能有些文章中所有的词都没超过它们的 TFIDF 均值的中位数, 调整后文章就成了空的。在下面的代码中, 第 1 句代码 `dtm[row_sums(dtm)>0, ]` `row_sums(dtm)` 为求出每篇文章的总词数, 如果某篇文章的总词数小于或等于零, 就将这篇文章删除。第 2 句代码使用

`summary(col_sums(dtm))`对语料库进行最后一次检查，看一下经过调整后每个词在语料库中出现的总次数的分布。

```
1 dtm <- dtm[row_sums(dtm)>0, ]  
2 summary(col_sums(dtm))
```

## 11.3 上帝有多少个色子：话题数量估计

下面要介绍主题模型了。理解主题模型的难点在于要具备数学知识，因为其中使用了随机过程。

### 11.3.1 通俗地说一遍话题模型

先确定一下研究对象的层次结构，即：语料库—文档—主题—词语。

LDA 是隐性语义分析 (LSA) 的一种，它的目的就是找出文档中隐含的 topics。每一篇文章的形成都可以看作上帝的一次游戏，他通过掷色子的方式形成一篇文章。上帝掷色子的过程是绝密的，只能看到他形成的文章。LDA 就是模拟上帝掷色子的过程来估计每个面的概率分布。

LDA 假设上帝这样掷色子：

(1) 上帝有两大坛色子，第一个坛子装的是 doc-topic 色子，每一个色子代表一篇文章，色子的每个面代表一个 topic 编号；第二个坛子装的是 topic-word 色子，每一个色子代表一个 topic，色子的每个面代表一个 word 编号。

(2) 上帝随机地从第二个坛子里独立地抽取了  $k$  个 topic-word 色子，编号为  $1 \sim k$ ，这一步确定了上帝生成的语料库中总共包含了多少个主题 (topic)。

(3) 每次生成一篇新的文档时，上帝先从第一个坛子中随机抽取一个 doc-topic 色子，然后重复如下过程生成文档中的词：

- ① 投掷这个 doc-topic 色子，得到一个 topic 编号  $z$ 。
- ② 选择  $k$  个 topic-word 色子中编号为  $z$  的色子，投掷这个色子，于是得到了一个词。

---

注：由于 topic 和 word 的抽取过程是相互独立的，上帝可以先每次抽取一个 doc-topic 色子，然后投掷这个色子，为每一个词确定 topic 编号  $z$ 。然后再按编号投掷 topic-word 色子，确定每一个词。

---

话题模型的训练过程大概是这样的：首先对语料库中的每一篇文章中的每一个词  $w$ ，随机地赋予一个 topic 编号  $z$ ，然后再按照算法 (Gibbs, CTM) 根据词频关联关系，重新计算每一个词归属于每个 topic 的概率分布 (统计与词  $w$  关系最近的那些词被分到哪个主题下，然后将该主题的概率设置得高一些)。这样经过几次迭代，某一个 topic 下会聚集越来越多的隐性关联词，一直迭代下去会发现算法收敛，这样就形成了 topic-word 共现频率矩阵，也就是 LDA 模型。

以上叙述只为了讲解通俗易懂，帮助读者尽快理解概念，实际过程还是要追究到数学中的。推

荐读者阅读相关主题模型资料：<http://ir.dlut.edu.cn/NewsShow.aspx?ID=264>。

### 11.3.2 主题数估计与交叉检验

建立 LDA 模型的第一个难点就是要估计出上帝从第二个坛子里拿出了几个色子，也就是语料库中共有多少个主题 ( $k$ )。可以使用复杂度的变化 (perplexity) 和对数似然值的变化 (loglikelihood) 进行估计。前者随着主题数的增加而递减，后者随着主题数的增加而递增。一般两者变化趋于平缓时的主题数即可作为估计的主题数量。

很明显，估计主题数本质上是一次模型比较。根据统计学的观点，“比较”需要统计指标的支撑，仅仅进行一次建模然后比较复杂度和对数似然值多少有一些片面。因此，至少需要比较不同主题数下模型复杂度均值和对数似然值均值的变化。

这里选择  $K$  层交叉检验 (K-fold Cross Validation)。所谓  $K$  层交叉检验，即将数据随机平均分成  $K$  组，每个子集均作为一次测试数据，其余的  $K-1$  组子集作为训练数据。这样就可以得到  $K$  个模型 (这些模型除了训练数据不同，其他原始参数都相同)，对应  $K$  次测试后，就得到同一评价指标的  $K$  个值。它们的平均数作为此次  $K$  层交叉检验下模型性能的比较指标。 $K$  层交叉检验可以有效地避免模型发生过拟合、欠拟合状态，其结果也更加符合统计思想。

下面来算一笔账，以 10 层交叉检验作为评价体系，假设有 5、10、20 共 3 个不同的备选主题数，从中选出最优的主题数，需要循环 30 次训练—测试的过程。所以，如果遇到大的数据集，还是将程序改成并行计算更加省时，要进行检验，首先要创造合理的数据分组方案，将数据分为  $K$  份 (尽量相等)。

- $K$  层交叉检验分组

```
1 library(plyr)
2 CVgroup <- function(k, datasize, seed) {
3   cvlist <- list()
4   set.seed(seed)
5   n <- rep(1:k, ceiling(datasize/k))[1:datasize]
6   temp <- sample(n, datasize)
7   x <- 1:k
8   dataseq <- 1:datasize
9   cvlist <- lapply(x, function(x) dataseq[temp==x])
10  return(cvlist)
11 }
12 k <- 10
13 datasize <- dtm$nrow
14 cvlist <- CVgroup(k = k, datasize = datasize, seed = 1206)
```

生成  $k$  层交叉检验分组的自编函数 CVgroup 已经在第 10 章详细讲解了，这里不再赘述。这里进行 10 层交叉检验，其中 datasize 为总文章数，即 dtm 中的行数，执行函数就可以得到分组表单 list。

一台机器的强大与否看它的解析能力，同样，将大目标分解为小目标，将大任务分解为小任务

也是个人能力强的一种表现。

简单地说，对于未知分布  $q$ ，复杂度的值越小，说明模型越好，而对于数似然值，则是越大越好。基于复杂度和对数似然值判断语料库中的主题数量，就是计算不同主题数量下的复杂度和对数似然值之间的变化，可以分为以下几步：

- (1) 选定一个主题数。
- (2) 选定算法 (CTM、Gibbs) 建模。
- (3) 计算模型复杂度和对数似然值。
- (4) 重复步骤 (2) ~ (3) 做  $K$  层交叉检验。

对于不同的主题数，都要重复以上步骤才能探究复杂度和对数似然值的变化。要完成这个任务，需要做两个循环（嵌套）（没有什么比在 R 里使用循环更加恐怖的事情了）。但循环是我们最熟知的逻辑方式，为了讲解方便，下面先用嵌套循环完成这件工作，后面再进行优化。

- 几个参数

```
TPnum = seq(10, 60, by = 10)
SEED = 1206
k <- 3
cross = k
```

在上面的代码中设置了几个参数，TPnum 是准备检验的主题数序列，共检验 10、20、30、40、50、60 个主题数，目的是测出语料库中共包含多少个主题。SEED 用于设定随机数种子，一个特定的种子可以产生一个特定的伪随机序列，其目的让使用相同随机数种子的人可重复本书的结果。因为经常需要取随机数，所以，同样的代码再运行一次结果就不一样了。为保证结果可复制，可以使用 set.seed 函数，只要两个人的随机种子相同，产生的随机数就是一样的。cross 表示对 10 层交叉检验进行一次削减，将原来 10 次循环换成了  $K$  次循环（视情况而定，因为建模太耗时，没必要严格按照  $K$  层交叉检验进行），也就是说，评价指标的均值并不是 10 次平均值，而是  $K$  次平均值， $K$  也可以改为 10。

是不是被下面的代码吓到了，不要怕，其实很简单（这句话是假的，应对复杂的问题肯定要用更加复杂的方案，别想找捷径）。这个函数总共分为 3 部分，用 3 个大括号分开，第 1 个大括号中是函数主体；第 2 个大括号中是对不同主题数目（kv）的循环；第 3 个大括号中是每个主题数目下进行的  $K$  层交叉检验，虽然  $K$  层表示能做  $K$  次检验，这里为了节省时间只做了 3 次检验。

- 主题数建模评估

```
1 library(topicmodels)
2 selectTP <- function(dtm, TPnum = TPnum, SEED = SEED, cvlist, cross = k) {
3   per_ctm <- NULL
4   log_ctm <- NULL
5   for (m in TPnum)
6   {
7     per <- NULL
8     loglik <- NULL
9     for (i in 1:cross)
```

```

10  {
11    cat("R 正在构建主题数为", m, "层次为", i, "的主题模型",
        as.character(as.POSIXlt(Sys.time(), "Asia/Shanghai")), "\n")
12    test <- cvlist[[i]]
13    train <- setdiff(1:dtm$nrow, test)
14    Gibbs <- LDA(dtm[train,], k = m, method = "Gibbs",
                  control = list(seed = SEED, burnin = 100, thin = 100, iter
= 1000))

15    per <- c(per, perplexity(Gibbs, newdata=dtm[test,]))
16    loglik <- c(loglik, logLik(Gibbs, newjiedata = dtm[test,]))
17  }
18  per_ctm <- rbind(per_ctm, per)
19  log_ctm <- rbind(log_ctm, loglik)
20 }
21 return(list(perplex = per_ctm, loglik = log_ctm))
22 }

```

selectP 函数包括的参数有文档词频矩阵 (dtm) 和主题数量 (TPnum)，这里设置了 5、10、20、30、40、50、60 共 7 个不同的主题数 (TPnum)。SEED 用于设置随机数种子。cvlist 用于指定  $k$  层交叉检验的测试数据和训练数据。cross 用于设定交叉检验做几次。这里设置进行  $k$  次，也就是说，对不同主题数求其模型的复杂度和对数似然值各  $K$  次（可以对  $K$  赋小于 10 并且大于 0 的任意整数）。下面将函数分解讲解。

```

1 per_ctm <- NULL
2 log_ctm <- NULL

```

上面这两句代码用于产生两个空对象，分别用它们存储复杂度和对数似然值，比如 per\_ctm 的第一行对应主题数为 5 时的复杂度，因为每个主题数目计算  $K$  次复杂度值，所以 per\_ctm 的第一行包括  $K$  个复杂度值。同样，log\_ctm 也是一样的结构。

首先函数依据主题数进行循环，因为预先设置了 7 个主题数目，所以第一层循环 7 次，然后在第二层循环开始之前创造了两个空向量 per 和 loglik，分别用来存储  $K$  次循环交叉检验产生的复杂度和对数似然值，这样第二层循环结束时，per 就是一个包含了  $K$  个复杂度值的向量，如下代码所示。

```

per <- NULL
loglik <- NULL

```

其实下面一句代码并不重要，只是为了告诉读者循环现在进行到哪里（相当于进度条）。cat 函数将成分组合在一起并输出，这样，在执行代码时，可以看到：“R 正在构建主题数为 60，层次为 3 的主题模型。2016-08-03 23:50:41。”这句话的意思就是 R 正在执行主题数为 60 的第 3 层交叉检验。后半句输出了系统时间。Sys.time 函数用来获取系统时间。as.POSIXlt 函数将系统时间转化为指定区域的时间，然后再经过 as.character 函数转化为字符，最后输出时用“\n”换行，否则多次输出会连在一起。

```

cat("R 正在构建主题数为", m, "层次为", i, "的主题模型",
    as.character(as.POSIXlt(Sys.time(), "Asia/Shanghai")), "\n")

```

下面两句代码比较重要，目的是拆分出测试数据 `test` 和训练数据 `train`。不要忘记 `cvlist` 里存储的内容，它将语料库中的文档编号随机平均分为 10 ( $K$ ) 份，`cvlist` 中的每个元素对应其中一份的编号。换句话说，可以将 `cvlist` 中的任意一个元素所存储的文档编号当作测试数据，其他 9 ( $K-1$ ) 份作为训练数据。

```
test <- cvlist[[i]]
train <- setdiff(1:dtm$nrow, test)
```

在上面的代码中，`test <- cvlist[[i]]` 将 `cvlist` 中的第  $i$  个元素所包含的文档编号作为测试数据。`setdiff` 这个函数很有意思，`setdiff(x, y)` 即求出在向量 `x` 中存在，但是在向量 `y` 中不存在的元素序列。`1:dtm$nrow` 即产生一个从 1 开始和语料库文档数等长的编号序列，而 `Setdiff` 函数从中去掉 `test` 中存在的元素，正好将剩下的作为训练数据。

下面的代码开始构建基于 Gibbs Sampling 的 LDA 模型。`dtm[train,]` 用于按 `train` 的文档编号抽取语料库 `dtm` 中相应的文档作为训练数据；`k` 用来设置主题数目；`method` 用于指定算法；`control` 是一个存储控制迭代的参数集；`seed` 用于设置随机数种子；`burnin` 用于丢弃对改变结果作用很小的陈旧迭代次数；`thin` 是在结果中保留的迭代次数，`iter` 为总的迭代次数。

`perplexity` 用于测试模型的复杂度，接着使用 `test` 筛选语料库中的测试数据，然后将结果与 `per` 向量合并再赋值给 `per`；`logLik` 用于测试新数据的对数似然值，然后将计算结果与 `loglik` 合并在一起再赋值给 `loglik`。到这里循环的第二层做完了，得到 `per` 和 `loglik` 两个向量，它们分别包括  $k$  (`cross`) 个值。

```
1 Gibbs <- LDA(dtm[train,], k = m, method = "Gibbs",
               control = list(seed = SEED, burnin = 100, thin = 100, iter =
1000))
2 per <- c(per, perplexity(Gibbs, newdata = dtm[test,]))
3 loglik <- c(loglik, logLik(Gibbs, newdata = dtm[test,]))
```

然后，将 `per` 和 `per_ctm` 按行粘贴在一起就得到了相应主题数的复杂度。同样，将 `loglik` 和 `log_ctm` 按行粘贴在一起，就得到了不同主题数下的对数似然值。这样函数的第一层也循环结束了。

```
1 per_ctm <- rbind(per_ctm, per)
2 log_ctm <- rbind(log_ctm, loglik)
```

最后，将 `per_ctm` 和 `log_ctm` 组成一个新的 `list` 作为函数的结果返回。执行下面语句就可以估计 LDA 模型主题数了。

```
1 system.time((perlog <- selectTP(dtm = dtm, TPnum = TPnum, SEED = SEED,
cvlist = cvlist, cross = cross)))
```

其实，建立 LDA 模型还有很多其他算法（如 VEM、VEM\_fixed、CTM），下面的代码集成了一些算法，即被注释的部分，感兴趣的读者可以尝试其中任何一种（记得在评估复杂度和对数似然值时，将模型名称 `Gibbs` 改为相应的模型名称），当然，经过修改后也可以进行多种算法的比较。

- 多种 LDA 算法集成

```
1 selectTP <- function(dtm, TPnum = TPnum, SEED = SEED, cvlist, cross = k) {
2   per_ctm <- NULL
3   log_ctm <- NULL
```



```

4   for (m in TPnum)
5   {
6     per <- NULL
7     loglik <- NULL
8     for (i in 1:cross)
9     {
10      cat("R 正在构建主题数为", m, "层次为", i, "的主题模型",
          as.character(as.POSIXlt(Sys.time(), "Asia/Shanghai")), "\n")
11      test <- cvlist[[i]]
12      train <- setdiff(1:dtm$nrow, test)

          # VEM <- LDA(dtm[train,], k = m, control = list(seed = SEED)),
          # VEM_fixed <- LDA(dtm[train,], k = m, control = list(estimate.alpha
= FALSE, seed = SEED)),

          # CTM <- CTM(dtm[train,], k = m, control = list(seed = SEED, var =
list(tol = 10^-4), em = list(tol = 10^-3)))
13      Gibbs <- LDA(dtm[train,], k = m, method = "Gibbs",
                    control = list(seed = SEED, burnin = 100, thin = 100, iter
= 1000))

14      per <- c(per, perplexity(Gibbs, newdata=dtm[test,]))
15      loglik <- c(loglik, logLik(Gibbs, newjiedata = dtm[test,]))
16    }
17    per_ctm <- rbind(per_ctm, per)
18    log_ctm <- rbind(log_ctm, loglik)
19  }
20  return(list(perplex = per_ctm, loglik = log_ctm))
21 }

```

在第 10 章中使用了 `mdply` 函数优化 R 中的循环，这里“故伎重演”。将上面带有循环的函数改为 `mdply` 函数。具体的代码就不讲解了，和第 10 章使用的方法一样，其中 *i* 代表交叉检验的分组号，*j* 代表主题数。

- 效率提升

```

1 library(plyr)
2 library(reshape2)
3 j <- seq(10, 60, by = 10)
4 i <- 1:k
5 i <- rep(i, times = length(j))
6 j <- rep(j, each = k)
7 x <- cbind(i, j)
8 cvtest <- function(i, j) {
9   test <- cvlist[[i]]
10  train <- setdiff(1:dtm$nrow, test)

```

```

11 Gibbs <- LDA(dtm[train,], k = j, method = "Gibbs",
               control = list(seed = 1206, burnin = 100, thin = 100, iter
= 1000))
12 per <- perplexity(Gibbs, newdata=dtm[test,])
13 loglik <- logLik(Gibbs, newjiedata = dtm[test,])
14 temp <- c(i, j, per, loglik)
15 }
16 system.time(perlog2 <- mdply(x, cvtest))

```

使用上面的代码效率有所提高，但是当你面对上千万条或者上亿条数据时，上面的方法能让你等到地老天荒，唯一的办法只能尝试使用并行计算完成，在第 14 章单独讲解。

### 11.3.3 如何使用复杂度、对数似然值确定主题数

根据复杂度和对数似然值并不能得出一个判断主题数的完美阈值，而是需要依靠需求和经验进行判断。每次提到用经验判断，我马上会联想到“不靠谱”等负面词汇，对科学体系产生怀疑。但是，统计的目的之一就是衡量并预测大量不确定的事物。可以将复杂度和对数似然值变化的拐点对应的主题数作为标准主题数，而拐点后面的复杂度和对数似然值的变化趋于平缓。观察拐点和趋势需要将数据可视化，因此，下面需要分别做复杂度、对数似然值与主题数的趋势图。

- 数据整理

```

1 mper <- apply(perlog[[1]],1,mean)
2 k <- c(TPnum)
3 perdf <- as.data.frame(cbind(perlog[[1]], mper, k), row.names = c(1:7))
4 names(perdf) <- c("fold1", "fold2", "fold3", "mper", "k")
5 mlog <- apply(perlog[[2]],1,mean)
6 logdf <- as.data.frame(cbind(perlog[[2]], mlog, k), row.names = c(1:7))
7 names(logdf) <- c("fold1", "fold2", "fold3", "mlog", "k")

```

在将数据可视化之前首先要对数据进行整理，上面的代码将计算结果存储成了一个 list 并赋值给了 perlog。List 中的第 1 个元素存储的是复杂度，第 2 个元素存储的是对数似然值，因为第 1 个元素的每行对应不同主题数的 3 个复杂度值，因此需要先求出它们的均值。其中第 1 行代码使用 apply 函数，第 1 个参数指定数据，数据可以是数据框、矩阵或者数组；第 2 个参数指定操作是针对数据框的行还是列，行用 1 表示，列用 2 表示；最后一个参数指定操作的函数，这里使用求均值函数 mean，连起来就是对数据框的每一行求均值。

第 2 行代码将之前设置的主题数赋值给向量 k，然后将每次测试的复杂度、复杂度均值、主题数按列绑定在一起，最后使用 as.data.frame 转化为数据框，并重新设置行编号。因为 R 数据框的行编号不允许出现重复值，原有的行编号都是 per，所以使用 row.names 参数重新设定。最后使用相同的方法处理对数似然值，处理完成后为了防止结果丢失，可以将结果输出为 csv 格式文件保存。

- 载入绘图包

```

1 require("extrafont")
2 loadfonts(device="win")

```

```

3 title_with_subtitle = function(title, subtitle = "") {
4   ggtitle(bquote(atop(.(title), atop(.(subtitle)))))
5 }

```

上面的代码和函数在第 3 章中非常详细地讲解过，这里不再赘述。

- 图前数据整理

```

1 perdf <- melt(perdf, id = "k", variable.name = "fold", value.name =
"perplexity" )
2 cols <- c(rgb(red = 0, green = 137, blue = 130, max = 255),
3           rgb(red = 253, green = 107, blue = 117, max = 255),
4           rgb(red = 150, green = 47, blue = 207, max = 255),
5           rgb(red = 254, green = 125, blue = 165, max = 255))

```

上面的代码将 wide 型数据表转为 long 型数据表。melt 函数在前面已经讲过，这里不再赘述。这里将 id 列设为主题数目列 k 列，然后将融合产生的两列数据命名为 fold 和 perplexity。最后设定了绘图将要用到的 4 种颜色，颜色由 rgb 函数设定，和我们常见的办公软件中的红、绿、蓝三原色系一样。

数据可视化可以帮助数据分析人员更加直接地掌握分析结果的要点，同时它也是通俗直观地将结果传达给受众的方式。正所谓“字不如表，表不如图”。更加详细的可视化内容可参看第 3 章内容。这里先介绍一下 ggplot2 包，在 R 众多的绘图包里，ggplot2 包是最专业的非互动平面绘图包。它的绘图原理比较像 Photoshop，即按层次一层层地将图表原件叠加而形成一幅完整的图。虽然在第 3 章中非常详尽地讲解了 ggplot2 包，但我认为，在 PPT 仍然是主流的商务沟通辅助工具的今天，无论怎么强调数据可视化都不为过。

ggplot 函数用来设定图表的基本参数，包括指定数据集并分配数据在图表中指代的具体对象（见如下代码），第 1 个参数用于指定数据集；而参数 aes 就比较复杂，它指定图表的坐标系或者映射关系，即 X 轴和 Y 轴分别由哪些数据映射。读者应该掌握一个常识：在视图中，图表对象的位置、大小、形状、颜色、方向等都是由数据形成的具体形象，它们与数据存在完整的映射关系。这里使用 fold 变量指代对象的颜色，即后面往上叠加的对象的颜色统统按照 aes 里指定的变量分配对象颜色。

接着使用 geom\_line 为图层添加线形形状，将其大小 size 设置为 1.5；然后使用 geom\_point 添加数据点，这样在原来的图层上就添加了两个主要对象（点和线）了。尽管图像的很多设置还是 ggplot2 包的默认绘图设置，但已经比某些软件制作的图表漂亮很多了。

```

1 library(ggplot2)
2 p <- ggplot(perdf, aes(x = k, y = perplexity, colour = fold)) +
  geom_line(size=1.5) +
  geom_point(size=5)

```

从函数的赋值方式可以看出 ggplot2 包的工作原理：叠加。在下面的代码中，scale\_colour\_manual 函数用来指定对象的颜色，上面已经预备了 4 种颜色，存放在 cols 里。values 指定 cols 为主图对象的颜色，颜色种类一定要和 aes 里映射颜色的变量下的因子水平相等。scale\_x\_continuous 函数用来指定 X 轴的一些特征，这里设置刻度按照主题数目 k 进行分割，即 X 轴的刻度按照什么方式分割。以此类推，如果要设置 Y 轴就用 scale\_y\_continuous 函数。

```
l p <- p + scale_colour_manual(values = cols) +  
  scale_x_continuous(breaks = perdf$k) +  
  title_with_subtitle("估计主题数", "复杂度变化趋于平缓的拐点") +  
  ylab("复杂度 (perplexity)") +  
  xlab("主题数")
```

一般图片都需要一个标题,标题要具有画龙点睛的作用。一般学术杂志会将图片标题独立摆放,而商务类型报告会在图片上摆放标题。使用 `title_with_subtitle` 函数可以为图片添加标题,标题可以分为主标题和副标题,它的第 1 个参数用于设置主标题,第 2 个参数用于设置副标题。

坐标轴的标签标注清晰是数据可视化的最低要求,否则受众无法理解图形具体的内容和量级。`ylab` 函数用于标注 Y 轴的名称,即 y label;同样, `xlab` 用于修改 X 轴的标签。到这里读者可能已经发现 `ggplot2` 包中的函数具有对称性,即每一个函数名都按照坐标轴对称,这个对称性能帮我们记忆函数。

主图已经制作完成了,下面对主图做一些修改,比如修改背景、字体、网格线、图例等,这些修改基本都可以使用 `theme` 函数完成。

首先使用 `theme_bw` 函数指定图片背景为白色以及网格线为黑色,它有两个参数:设置字体大小的 `base_size` 和设置字体类型的 `base_family`,如果后面不进行针对性的修改,那么所有字体都会按照这个函数的设置来执行。

`theme` 函数是一个百宝箱,几乎所有有关图片美化等具有共性的设置都可以放到这个函数里。`plot.margin` 参数用来设置图片四边的留白大小(下、左、上、右),其有两个参数,一个用于设置数值(`unit`),另一个用于设置数值单位。单位有 `lines` 和 `inches` 之分,一个单位的 `lines` (`lines of text`) 相当于 0.2 inches。

`panel.background` 参数用于设置画板背景的颜色,注意是画板的背景,不是整张图片的背景。`Ggplot` 函数里的元素(`element`)分为 `rectangle`(形状)、`text`(文本)和 `line`(线)等,背景属于形状,因此这里使用 `element_rect` 函数,其中参数 `fill` 用来设置填充色,`plot.background` 用来设置整张图片的背景色。

`plot.title` 用于设定图表标题,因为标题属于 `text`(文本),所以使用 `element_text` 函数,参数包括大小(`size`)、字体类型(`family`)、粗体(`face`)、水平位置调整(`hjust`)、垂直位置调整(`vjust`)以及颜色(`colour`)。

```
p <- p + theme_bw(18) +  
  theme(plot.margin = unit(c(1, 1, 1.25, 0.5), "lines"),  
    panel.background = element_rect(fill=rgb(red = 242, green = 242, blue  
= 242, max = 255)),  
    plot.background = element_rect(fill=rgb(red = 242, green = 242, blue  
= 242, max = 255)),  
    plot.title = element_text(size = rel(1.2), family = 'STXingkai',  
      face = 'bold', hjust = -0.05,  
      vjust = 1.5, colour = '#3B3B3B'),  
    panel.grid.major = element_line(colour=rgb(red = 146, green = 146, blue
```

```

= 146, max = 255),size=.75),
  panel.border = element_rect(colour=rgb(red = 242, green = 242, blue =
242, max = 255)),
  axis.ticks = element_blank(),
  axis.text.x = element_text(colour="grey20", size=12),
  axis.text.y = element_text(colour="grey20",size=12),
  axis.title.y = element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=.5),
  axis.title.x=element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=-.5),
  legend.position="bottom",
  legend.title=element_blank(),
  legend.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255), size=.1),
  legend.text = element_text(size = 10, face = "bold"))

```

在上面的代码中，`panel.grid.major` 用来设置网格线的颜色，网格线分为主网格线和次网格线，这里设置的是主网格线（`grid.major`）的颜色。`panel` 用来指代网格线归属，即主网格线归属于画板。`panel.border` 用来设置画板边缘的颜色，`axis.ticks` 指一定坐标轴的刻度线，`element_blank` 表示不使用刻度线。

接下来是 4 对对称参数，`axis.text` 参数用来设定对称轴的刻度线标签。因为标签属于文本，所以使用 `element_text` 函数，这里仅仅设置了 X 轴、Y 轴刻度标签的字体颜色和大小，当然你也可以设置其类型、方向等。`axis.title` 是另外一对对称参数，用来调整数轴的名称，也是用 `element_text` 函数设定，即对 `xlab`、`ylab` 设定的内容有针对性美化。

接着使用 `legend.position` 参数将图例放在底部。另外，`legend.title` 参数将图例标题设置为空，这时图例和图片背景不能完美地融合在一起，所以需要使用 `legend.background` 修改图例背景色，使用 `legend.text` 修改图例标签的字体大小和粗细。

```

1 png(filename = "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型是很多牛
人过不去的坎儿/plot/perplex.png", width = 860, height = 560)
2 print(p)
3 dev.off()

```

基本上复杂度可视化就完工了，然后将图表保存。使用 `png` 函数开启 `png` 设备，需要指定图片保存的完整地址，还要设置图片的长、宽比例。然后使用 `print` 函数打印图片，最后关闭设备，这样就将图片保存了，如图 11-1 所示。

根据复杂度的变化曲线，可以判断文本中大概包括 20 个话题。对于这个数字，可以通过对数似然值进一步确认。在绘制对数似然值的变化曲线时，因为存储对数似然值的 `logdf` 与复杂度的结构一样，所以在绘制时可以采用相同的方法和函数。

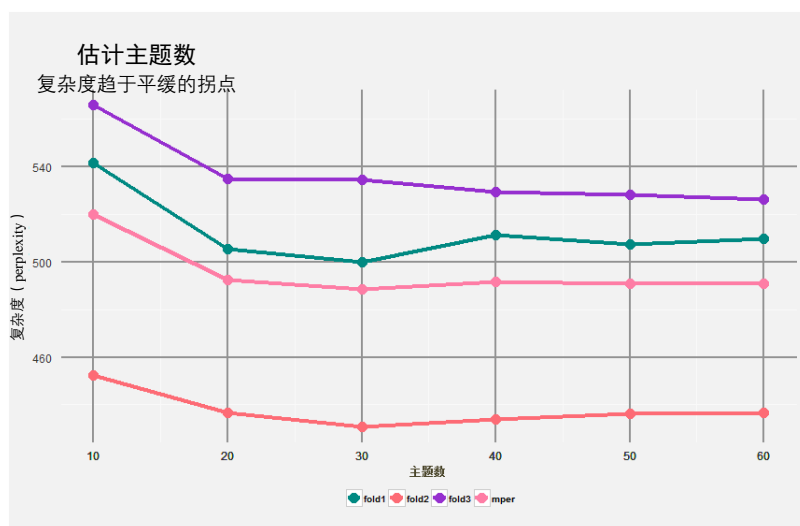


图 11-1

- 绘制对数似然值变化曲线

```
1 logdf <- melt(logdf, id = "k", variable.name = "fold", value.name = "loglik" )
2 p <- ggplot(logdf,aes(x = k, y = loglik, colour = fold)) +
  geom_line(size=1.5) +
  geom_point(size=5) +
  scale_colour_manual(values = cols) +
  scale_x_continuous(breaks = logdf$k) +
  title_with_subtitle("估计主题数","对数似然值变化趋于平缓的拐点") +
  ylab("对数似然值 (logLik) ") +
  xlab("主题数") +
  theme_bw(18) +
  theme(plot.margin = unit(c(1, 1, 1.25, 0.5), "lines"),
        panel.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
        plot.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255)),
        plot.title = element_text(size = rel(1.2), family = 'STXingkai' ,
                                face = 'bold', hjust = -0.05,
                                vjust = 1.5, colour = '#3B3B3B'),
        panel.grid.major = element_line(colour=rgb(red = 146, green = 146, blue
= 146, max = 255),size=.75),
        panel.border = element_rect(colour=rgb(red = 242, green = 242, blue =
242, max = 255)),
        axis.ticks = element_blank(),
        axis.text.x = element_text(colour="grey20", size=12),
```

```

axis.text.y = element_text(colour="grey20",size=12),
axis.title.y = element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=.5),
axis.title.x=element_text(size=11,colour=rgb(red = 74, green = 69,
blue = 42, max = 255),face="bold",vjust=-.5),
legend.position="bottom",
legend.title=element_blank(),
legend.background = element_rect(fill=rgb(red = 242, green = 242, blue
= 242, max = 255), size=.1),
legend.text = element_text(size = 10, face = "bold"))

```

仅仅需要修改一下数据、图表标题、X 轴、Y 轴标签，即可将绘制复杂度折线图的代码修改为绘制对数似然值的代码，然后将图片命名并保存。其实 ggplot 函数的 theme 部分可以存储起来备用，因为形成个人绘图风格后，theme 部分就不会发生太大变化了。

```

1 png(filename = "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：很多牛人
2 print(p)
3 dev.off()

```

依据复杂度变化曲线和对数似然值变化曲线，可以判定 47 份文本中大概可以构建 20 个话题，下面不妨就以 20 为话题数建立 LDA 模型，如图 11-2 所示。

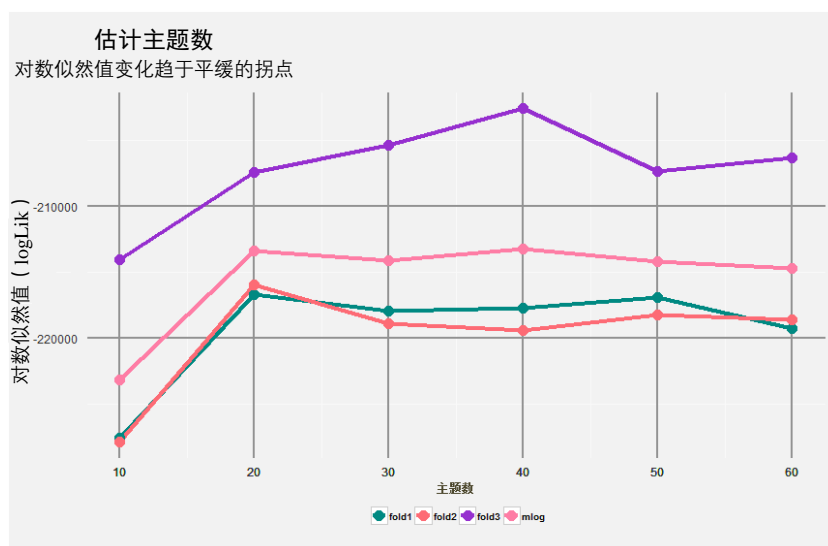


图 11-2

## 11.4 LDA 话题模型竟然能输出这么多关系

话题模型能做什么工作？可能直到看到这里，很多读者都没有一个清晰的答案。话题模型完成的是文本分类任务。那么根据话题模型我们能得到哪些有用的结果呢？话题模型能够初步估计出语料（文本）中有多少个话题，或者说大家谈论了多少个话题；能判断文本的话题归属，即给定一个文本能够判断文本可以归属到哪些话题；可以输出话题的关键词；判断两个文本之间的相似性。如果熟悉话题模型的原理，则可以尽可能地和业务问题联系在一起。

### 11.4.1 输出主题——词汇及其概率矩阵

首先，需要根据前面估计的 20 个主题重新构建模型，然后我们就可以了解每一个主题下包含哪些关键词及归属的概率矩阵了。

- 构建 LDA 模型

```
1 m <- 20
2 SEED <- 1206
3 ldalist <- list(
4   VEM = LDA(dtm, k = m, control = list(seed = SEED)),
5   VEM_fixed = LDA(dtm, k = m, control = list(estimate.alpha = FALSE, seed = SEED)),
6   Gibbs = LDA(dtm, k = m, method = "Gibbs",
7               control = list(seed = SEED, burnin = 100, thin = 100, iter = 1000)),
8   CTM = CTM(dtm, k = m,
9             control = list(seed = SEED, var = list(tol = 10^-4), em = list(tol = 10^-3))) )
save(ldalist, file = paste("H:/zimeiti", "/ldalist.Rdata", sep = ""))
```

上面的代码使用不同的算法依次建立了 4 个模型，第 1 个模型基于 VEM 算法，第 2 个模型基于修正后的 VEM 算法，第 3 个模型基于 Gibbs Sampling 过程，第 4 个模型基于 CTM 算法，然后将模型结果存放在名字为 ldalist 的 R 文件里，最后将结果保存到硬盘中备用。

当然，读者也可以根据自身需要构建其中一种模型。

- Gibbs Sampling

```
1 k = 20
2 SEED <- 2001
3 Gibbs <- LDA(dtm, k = k, method = "Gibbs",
               control = list(seed = SEED, burnin = 100, thin = 100, iter = 1000))
```

LDA 模型终于搭建完成了，之后就要使用了。就模型内现有的结果而言，首先可以查看每个话题包括哪些词汇，然后可以查看每个文档的话题归属。如果对一篇文章输出一个话题不满足，那么也可以查看每一篇文章的文本话题概率矩阵，另外，还可以测试新数据的话题归属。



- 查看话题词汇

```
1 termsgibbs <- terms(Gibbs, 15)
2 termsgibbs <- as.data.frame(termsgibbs, stringsAsFactors = FALSE)
3 write.csv(termsgibbs, "H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：
很多牛人过不去的坎儿/plot/termsgibbs.csv")
```

使用 `terms` 函数可以获得主题词汇，在上面的代码中仅提取前 15 个主题词汇，`term` 输出的对象为 `array`，需要将其转化为数据框，然后输出 `csv` 文档。当然，每一个词归属到某一主题下的可能性都不同，这就涉及概率值，所以，必要时还需要连带输出主题下的词汇及其概率。

- 输出主题——词汇概率图

```
1 terms <- as.data.frame(t(posterior(Gibbs)$terms))
2 names(terms) <- paste("Topic", c(1:k), sep = ".")
3 terms <- transform(terms, term = rownames(terms))
4 topiccihui <- melt(terms, id = "term", value.name = "prob", variable.name
= "topic")
5 names(topiccihui) <- c("TERM", "TOPIC", "PROB")
```

在上面的代码中，第 1 行代码获得主题——词汇的概率矩阵，但是它的行名称是词汇，列名称是主题编号，需要将其转化为 3 列数据：词汇、主题编号、概率。所以，第 2 行代码重命名 `terms` 的列，给每一个主题编号前面添加英文的“Topic.”。第 3 行代码使用 `transform` 函数在 `terms` 数据框中添加新列 `term`，并将行名称（即词汇）赋值给新列。第 4 行代码使用 `melt` 函数以 `term` 为 `id` 列进行长表与宽表的转换。第 5 行代码重新命名长表。

## 11.4.2 输出主题——文档归属及其概率矩阵

在前面我们查看了主题与词汇的关系，即每个主题相关的关键词及其概率矩阵，大概了解了每个主题的内容，那么文档和主题有什么关系呢？我们首先要知道语料库中每篇文档与主题的对应关系。

- 查看文档主题归属

```
1 doctopic <- topics(Gibbs)
2 temp <- names(doctopic)
3 temp <- cbind(as.numeric(temp), as.numeric(doctopic))
4 doctopic <- as.data.frame(temp)
```

在上面的代码中，`topics` 函数输出主题模型中每一篇文章的主题归属，输出结果是一个向量，每一个元素的名称为文本编号，数值为主题编号。接着使用 `names` 函数提取文本编号，然后将文本编号转化为数值后按列 `cbind` 排在一起，最后转化为数据框输出。

- 查看文档主题概率矩阵

```
1 gammadf <- as.data.frame(Gibbs@gamma)#提取主题文档矩阵,输出行为文档,列为 topic
的后验概率矩阵表
2 names(gammadf) <- c(1:k)
```

在上面的代码中，`Gibbs@gamma` 用于提取文本主题概率矩阵，将其转化为数据框后，数据框的

行代表文本编号，列代表主题编号，每个单元格就是文本归属于某一个主题的概率。

- 为新文本划分主题

```
1 newdoctopics <- posterior(Gibbs,newdtm )
```

在上面的代码中，对新的文本进行清洗、分词、调整等工序后，形成新的文本词频矩阵，比如取名为 newdtm，然后使用 posterior 函数就可以预测新文本的主题概率分布。posterior 函数的其中一个参数为已经建立的主题模型，另一个参数为新数据的文本词频矩阵，也就是说，LDA 模型也具有预测新文本主题归属的能力。

### 11.5 话题之间也有社交（衍生）关系吗

在 11.4 节中，将 47 年文案文本集划分成 20 个话题，我们也基本了解了这些话题包含的关键词和文档的话题归属，但是话题之间的关系还没有研究。下面不妨这样定义话题之间的关系：如果两个话题之间存在一个共有关键词，则认为话题之间产生一次关系。不要小看定义或者概念，因为它们通常是解决问题的捷径，这也是所谓的“挖掘之道”。概念和指标一样，是划定界限的标准，读者一定要尝试构建自己的指标和概念体系。

- 数据准备

```
1 topicterms <- terms(Gibbs, 10)
2 topicterms <- as.data.frame(topicterms, stringsAsFactors = F)
```

在上面的代码中，首先需要提取每个话题下的关键词，这里仅提取每个话题下前 10 个关键词。其中第 1 行代码提取关键词。第 2 行代码将关键词话题矩阵转化为数据框。

要研究网络关系，再没有比绘制网络关系图更加直观的方式了，这里所说的网络关系图与第 9 章讲述的网络关系图稍有区别。在第 9 章中，讲述了国家、演员的社交网络，而这里讲的是主题的关系网络，主题是由词汇组成的。首先，我们应该将同一主题下的词汇连接起来，形成一个完整的话题词汇链条，然后再通过词汇之间的关系将话题连接起来，如主题 1 应该有来如下所示的词汇链：

"一九五八年"--"纲要"--"运动"--"一九五七年"--"人民公社"--"战线"--"一九五九年"--"基本建设"--"跃进"--"超额"

在绘制网络关系图之前，需要将数据转化为 igraph 对象，这就要求数据是一个两列的数据框。在第 9 章中，为了生成符合要求的数据框，使用了 combn 函数，但是为了让同一个话题下的词汇形成如上述的词汇链，这里需要将主题 1 的元素摆放成如下所示的形式：

```
1      [,1]      [,2]
2 [1,] "纲要"      "一九五八年"
3 [2,] "运动"      "纲要"
4 [3,] "一九五七年" "运动"
5 [4,] "人民公社"   "一九五七年"
6 [5,] "战线"      "人民公社"
7 [6,] "一九五九年" "战线"
```

```

8 [7,] "基本建设"    "一九五九年"
9 [8,] "跃进"        "基本建设"
10 [9,] "超额"        "跃进"

```

即同一主题下的词汇不能进行随机组合，只能按照 1 和 2、2 和 3、3 和 4 这样有序的关系组合，embed 函数刚好可以将向量转化为这样的组合，其中第 1 个参数指定向量，第 2 个参数指定讲向量转化为几列有序组合。

- 数据准备

```

1 adjacentlist <- lapply(1:20, function(x) embed(topicterms[, x], 2)[, 2:1])
2 edgelist <- as.data.frame(do.call(rbind, adjacentlist), stringsAsFactors
=F)
3 topic <- unlist(lapply(1:20, function(x) rep(x, 9)))
4 edgelist$topic <- topic

```

在上面的代码中，第 1 行代码中的 topicterms[, x] 表示抽取编号为 x 的主题下的所有词汇，然后 embed(topicterms[, x], 2) 将词汇转化成两列有序组合，每个主题刚好包括 9 行数据；embed(topicterms[, x], 2)[, 2:1] 表示将产生的组合第 1 列和第 2 列调换位置，使用 lapply 函数将 20 个主题的词汇均按照上述设置做一次转换；产生的结果是 list。第 2 行代码将 list 的元素按行捆绑为数据框。第 3 行代码将 20 个主题编号每个重复 9 次，并将结果 unlist 成为 vector 向量。第 4 行代码将 topic 赋值给数据框 edgelist 的新列 topic，可被 graph.data.frame 函数转化为 igraph 对象的数据框就制备好了。

- 生成主题关系网络

```

1 library(igraph)
2 topicnet <- graph.data.frame(edgelist, directed = T )
3 layout <- layout_fruchterman_reingold(topicnet)
4 V(topicnet)$size <- degree(topicnet)/3
5 temp <- sample(colors()[26:137], 20)
6 E(topicnet)$color <- unlist(lapply(temp, function(x) rep(x, 9)))

```

在上面的代码中，第 2 行代码将 edgelist 转化为 igraph 对象，这里构建的网络关系是指向型的，所以 directed 参数设置为真。第 3 行代码使用 layout\_fruchterman\_reingold 函数计算网络的布局，这些函数都已经在第 9 章介绍过了。前面讲过计算节点度数的方法，第 4 行代码直接使用 degree 函数统计节点的度数并赋值给节点的 size 属性；在终端执行 colors 函数时可以看到它存储了 600 多种颜色的名称。第 5 行代码使用 sample 函数随机抽取其中的 20 种颜色备用。第 6 行代码使用 lapply 函数将 20 种颜色每个重复复制 9 次，然后 unlist 为 vector 赋值给边线的颜色，其目的是将同一主题的词汇用相同颜色的边线连接在一起。

下面就可以将网络关系绘制出来了。

- 绘制主题关系网络图

```

1 png(paste("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：很多牛人过不去
的坎儿/data/plot", "/topic_graph_gibbs.png", sep=""),
2     width = 800, height = 800)
3 par(mai=c(0,0,0,0), bg = rgb(242, 242, 242, max = 255))
4 plot(topicnet, layout = layout,

```

```
5     vertex.color = rgb(red = 254, green = 67, blue = 101, max = 255),#  
设置节点颜色  
6     vertex.frame.color = NA,#去除节点边缘线  
7     vertex.label.cex = 0.9, #设置字体大小  
8     vertex.label.color = rgb(0, 0, 0, max = 255, alpha = 150),  
9     vertex.label.font = 1,#设置字体粗细  
10    vertex.label.dist = 0.3,  
11    edge.curved = TRUE, #设置边缘线类型  
12    edge.width = 2,#设置边缘线宽度  
13    edge.arrow.size = 0.05)#设置箭头大小  
14 dev.off()
```

在上面的代码中，第 1 行代码启动 png 设备，并使用 paste 函数将图文件名和路径连接为完整的存放路径，这里仅仅使用了 plot 函数里的 edge.curved 参数用于指定边线的类型为曲线。

## 11.6 话题模型的几个强大衍生品

以上我们构建了话题模型，也介绍了一些应用，了解了怎样通过话题考查语料库的话题中心的转移，但是显然不能止步于此，话题模型还能帮助我们做什么呢？下面利用话题模型的输出结果解读更多的内容。要想有好的研究成果，首先要提出有意义的问题，这里不妨提出如下问题：根据文本之间的相似性，能否将文本的发展划分为几个阶段，各个阶段具有什么样的主题？

**漂亮的研究结果善始于建设性的问题。**

### 11.6.1 话题模型提取特征词

文本聚类无非是将文本中的信息作为衡量差异的变量进行聚类分析，在这里我们使用主题模型提取特征词，然后统计每一篇文章中特征词出现的频率，作为特征变量进行层次聚类。

- 特征词提取

```
1 topicterms <- terms(Gibbs, 20)  
2 topicterms <- as.vector(topicterms)  
3 topicterms <- gsub("\\\\n", "", topicterms)  
4 msgWords <- msgWords[msgWords$Term %in% topicterms,]  
5 # Logic <- rep(1, length(msgWords[,1]))  
6 # msgWords <- as.data.frame(cbind(msgWords, Logic), stringsAsFactors = F)  
7 msgTerm <- dcast(msgWords, Docid ~ Term, value.var = "Logic")  
8 rownames(msgTerm) <- msgTerm[,1]  
9 msgTerm <- msgTerm[,-1]
```

在上面的代码中，第 1 行代码用于提取话题模型中每个主题下的前 20 个词汇，作为特征词。第 2 行代码将上一步提取的主题词汇矩阵转化为向量。因为 tm 包对中文支持不好，在有些词汇中间会

无故加了一个换行符 “\n”，例如 “高新技术\n 产业”，所以第 3 行代码要将这个换行符去掉。因为 “\” 在正则表达式中具有特殊意义，所以这里需要添加一个 “\” 作为转义字符才能匹配，即用 `gsub` 函数替换所有的换行符为空。第 4~6 行代码为了统计每一篇文章关键词的词频，构建文档词频矩阵。第 4 行代码生成一个统计词频的辅助变量。第 5 行代码将辅助列和原来的数据框捆绑在一起（因为之前的操作已经添加了辅助列，这里把它注释掉了）。第 7 行代码用 `dcast` 函数将长表变宽表，统计词频。在 R 中做聚类时，需要将 `case` 编号列赋值给 `row.names`，然后删除原来的 `case` 编号列，所以第 8~9 行代码用来完成这一步。如果你在做聚类或者计算相似性、距离时，系统无端报错，但是又找不到原因，那么基本上是这个原因了。

## 11.6.2 三种方法确定聚类的类数和文本层次聚类

前面已经整理好了用于文本聚类的数据，其实就聚类而言，大多数人第一个想到的方法是分群、分类问题，但是在实际应用中，方法可真是五花八门。比如在挑选训练数据集时，可以先聚类，让算法完成初始的粗分，虽然不能完全解决挑选训练数据集的问题，但是至少能减少人力，达到事半功倍的效果。

怎么确定聚类的类数？这是一个很无厘头的问题，但也是一个必须要解决的问题。因为在自动化程序下，不可能人工干预程序，总要有方法或指标让程序自动预估一下聚类的数据量，幸好确实存在一些预估方案。

- 方法 1：中心化分法（PAM）

可以使用中心划分法确定聚类的数量，如下代码所示。

```
msgTerm <- scale(msgTerm)
library(fpc)
library(cluster)
pamk.best <- pamk(msgTerm)
cat("通过中心化分法估计的最佳聚类数:", pamk.best$nc, "\n")
```

一般在聚类之前要对数值型数据进行一次标准化。PAM（Partitioning Around Medoids）是一种基于  $K$ -中心点的聚类算法，它和  $K$ -Means 算法有非常明显的区别，其准确性比较高，但是计算中心点的效率比较低。所谓效率低，就是面对大数据时计算速度比较慢。上面的代码使用 `fpc` 包的 `pamk` 函数计算最佳的类数；使用 `cat` 函数打印出存储在 `pamk.best$nc` 中的最佳聚类数。

- 方法 2：Calinski-Harabasz 方法

误差项平方和（SSE）通过类内方差衡量一个聚类的聚合性。聚类效果的度量一般应该从两个方面衡量：类内和类间，最优的聚类效果应该具有最小的类内距离和最大的类间距离。换句话说，最优的聚类应该具有最小的类内聚合度和最大的类间分离度。类群的聚合度（cohesion）度量类中样本之间的相似程度，而类的分离度（separation）度量类之间的相异程度。好的聚类结果应该既具有较小的类内聚合度，同时又具有较大的类间分离度。

CH（Calinski-Harabasz）系数就是基于类内聚合度和类间分离度定义的聚类评价指标，对于聚

类问题，类间分离度越大，类内聚合度越小，则 CH 系数的值就会越大，说明这样的划分越优秀。

```
1 require(vegan)
2 fit <- cascadeKM(msgTerm, 1, 10, iter = 1000)
3 calinski.best <- as.numeric(which.max(fit$results[2, ]))
4 cat("Calinski-Harabasz 系数估计的最佳聚类数:", calinski.best, "\n")
```

在上面的代码中，第 1 行代码加载 `vegan` 包。第 2 行代码使用 `cascadeKM` 基于 Kmeans 方法计算 CH 值，第 1 个参数指定最小聚类数，第 2 个参数指定最大聚类数，`iter` 参数指定迭代次数。第 3 行代码使用 `which.max` 在上一步的结果中提取 CH 值最大聚类数。使用上述方法可以很容易看出 CH 值变化情况，直接选出最大 CH 值所对应的聚类数即可，但是这个方法无法应对较大数据量。

- 方法 3：轮廓系数

当然你也可以使用轮廓系数，它结合内聚度和分离度两种因素，即同时考察了组内相似性和组间差异性，可见轮廓系数的值是介于  $[-1, 1]$ ，越趋近于 1，代表内聚度和分离度都相对较优，我比较习惯使用这个指标。

```
1 library(fpc)
2 asw <- numeric(20)
3 for (k in 2:20)
4 asw[[k]] <- pam(msgTerm, k) $ silinfo $ avg.width
5 k.best <- which.max(asw)
6 cat("通过轮廓系数 (silhouette) 估计的最佳聚类数:", k.best, "\n")
```

在上面的代码中，轮廓系数也是通过 PAM 算法完成的。第 3 行代码创建了一个空的数值类型向量，并设置向量长度。第 4 行代码使用 `pam` 函数对每一种聚类数进行一次聚类并使用 `$` 符号提取轮廓系数，轮廓系数存储在建模结果的 `silinfo` 元素内的 `avg.width` 元素中。第 5 行代码是找出轮廓系数最大的聚类数。

如果你想提高代码的运行效率，则可以尝试使用 `lapply` 家族或者并行计算提高速度。说实话，划分聚类数真的是一件仁者见仁，智者见智的事情。就像切西瓜，切几瓣是很随意的事情，无论你的划分方法多么科学，如果客户不认可，那么一切都是没用的。

- 文本层次聚类

```
1 msgTerm <- scale(msgTerm)
2 d <- dist(msgTerm, method = "euclidean")
3 fit <- hclust(d, method="ward.D")
```

层次聚类算法是将样本点自底向上合并成一棵树或者自顶向下分裂成一棵树的过程，这两种方法被称为凝聚和分裂。但无论使用哪种方法，都需要一种评价样本之间相似性的指标。通常使用距离来衡量样本之间的相似性。在上面的代码中，第 1 行代码对数据标准化。第 2 行代码使用 `dist` 函数完成距离计算，其中 `method` 参数用于选择距离计算的算法。这里使用欧式距离，至于计算距离的方法可以查看 R 的帮助文档。除了要了解算法，还要明白这些方法对业务的意义和适用的数据类型。第 3 行代码使用 `hclust` 函数完成文本层次聚类，其中 `method` 参数指定聚类的方法，有类平均法 ( `average` )、中间距离法 ( `median` )、最短距离法 ( `single` )、最长距离法 ( `complete` ) 等，这里使用离差平方和法 ( `ward` )，这样层次聚类就完成了，下面要对聚类结果进行可视化。

### 11.6.3 漂亮的文本聚类树和批量绘制大类词云图

前面对文本进行了层次聚类，并将文本分为了 4 大类。下面就将层次聚类树可视化，同时批量绘制词云图看一看每一个类群大概说了什么。

- 数据准备

```
1 library(ggdendro)
2 dendr <- dendro_data(fit, type = "rectangle")
3 clust <- cutree(fit, k = 4)
4 clust.df <- data.frame(label = names(clust), cluster = factor(clust))
5 dendr[["labels"]] <- merge(dendr[["labels"]], clust.df, by="label")
```

在上面的代码中，第 1 行代码载入 ggdendro 包。第 2 行代码专门使用 dendro\_data 函数提取聚类模型中一些用于绘图的结果数据，由于聚类树状图既不是线也不是点，所以将 type 定为了 rectangle。第 3 行代码使用 cutree 函数将层次聚类切分为几个大类，尽管上面的算法帮助我们确定的类数为 2 类或 10 类，但是层次聚类可能不适用上面的评估结果，这里暂时定位 4 个大类。第 4 行代码将分类结果转化为数据框，使用 names 函数提取每篇文本的名称，使用 factor 函数提取每篇文本被分到的大类编号。第 5 行代码将 clust.df 与 dendr 的 labels 元素按照它们都包含的 label 列，关联在一起，可以通过 head(dendr[["labels"]]) 查看关联后的变化。

- 设定绘图风格

```
1 cols <- c(rgb(red = 0, green = 130, blue = 137, max = 255),
            rgb(red = 71, green = 71, blue = 71, max = 255),
            rgb(red = 61, green = 95, blue = 241, max = 255),
            rgb(red = 254, green = 67, blue = 101, max = 255))
theme <- theme(axis.line.y = element_blank(),
               axis.ticks.y = element_blank(),
               axis.text.y = element_blank(),
               axis.title.y = element_blank(),
               panel.background = element_rect(fill =
                                               rgb(red = 242, green = 242, blue
= 242, max = 255)),
               plot.background = element_rect(fill =
                                               rgb(red = 242, green = 242, blue
= 242, max = 255)),
               plot.title = element_text(size = rel(2), family = 'Times New
Roman' , face = 'bold', hjust = 0.2, vjust = 0.5, colour = '#3B3B3B'),
               panel.grid.major = element_blank(),
               legend.position = "none")
```

上面的代码为 4 种分类类型准备 4 种标度颜色，绘图的主题 theme 是不变的。

- 绘制层次聚类树

```
1 p <- ggplot() +
2   geom_segment(data = segment(dendr), aes(x = x, y = y, xend = xend, yend
= yend)) +
```

```
3   geom_text(data=label(dendr), aes(x, y, label=label, hjust=0, color =  
cluster),  
4       size=5) +  
5   scale_colour_manual(values = cols) +  
6   coord_flip() + scale_y_reverse(expand=c(0.2, 0)) +  
7   title_with_subtitle("") +  
8   ylab("") + theme  
9   print(p)
```

- 保存绘图结果

```
1 png(filename = paste("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：  
很多牛人过不去的坎儿/plot/zhengfucluster_50_", ".png", sep = ''),  
2   width = 680, height = 680)  
3 print(p)  
4 dev.off()
```

结果如图 11-3 所示。

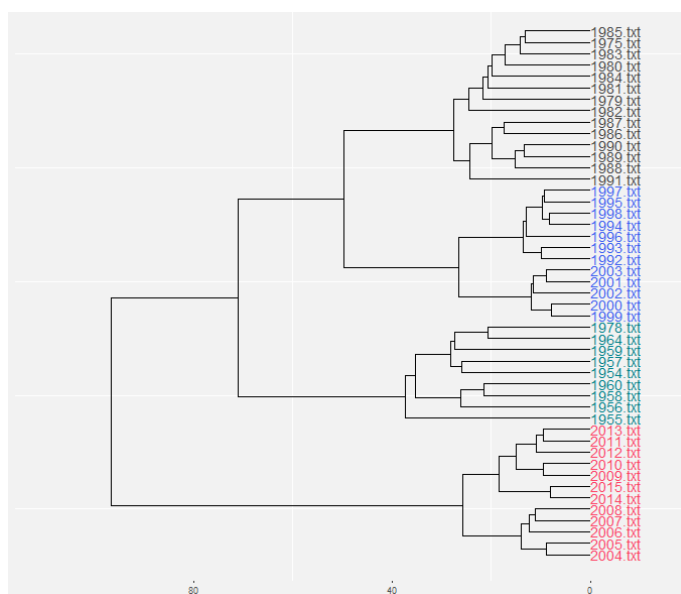


图 11-3

可以看到，我们根据文本的词汇将文本集分成了 4 个鲜明的类别。那么每个类别都讲了哪些内容呢？我们根据聚类划分阶段，绘制不同阶段的词云，具体过程如下所示。

- 数据准备

```
1 msgWords <- msgWords[msgWords$Term %in% topicterms,]  
2 msgTerm <- dcast(msgWords, Term ~ Docid, value.var = "Logic")  
3 msgTerm <- melt(msgTerm, id = 1)  
4 msgTerm <- msgTerm[-which(msgTerm$value == 0),]
```



```
5 names(clust.df) <- c("variable", "cluster")
6 msgTerm <- join(msgTerm, clust.df)
```

msgWords 存储了每一篇文章的分词结果，在上面的代码中，第 1 行代码筛选出主题模型提取的特征词。第 2 行代码使用 dcast 函数统计每篇文章中每一个特征词发生的频率，这是一个长表变宽表的过程。第 3 行代码将 Term 列作为 id 将原来的宽表变为长表。第 4 行代码删除词频为零的记录。第 5 行代码对 clust.df 重新命名，保持文章名称列和 msgTerm 中的文章名称列一样，同为 variable。第 6 行代码使用 join 函数将两个数据框按照 variable 进行左关联，这样每篇文章就关联上了聚类的分类结果。

- 批量绘制词云图

```
1 library(wordcloud)
2 library(RColorBrewer)
3 ciyun <- function(i) {
4   temp <- msgTerm[msgTerm$cluster == i,]
5   temp <- aggregate(value ~ Term, data = temp, sum)
6   temp <- temp[order(-temp$value), ]
7   colors <- brewer.pal(8, "Dark2")
8   png(filename = paste("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 11 章话题模型：
很多牛人过不去的坎儿/plot", i, ".png", sep = ''), width = 600, height = 600)
9   par(bg = rgb(red = 242, green = 242, blue = 242, max = 255))
10  wordcloud(words = temp$Term, freq = temp$value, scale = c(6, 0.9),
11            min.freq = mean(temp$value),
12            max.words = 200, colors = colors, random.order = F, random.color
= T, rot.per = 0.5, font = 2, family = "serif")
13  dev.off()}
14 lapply(1:4, function(i) ciyun(i))
```

上面写了一个批量绘制词云的函数，其中 i 用于指定绘制哪个大类的文本词云，其中第 4 行代码筛出所有和大类 i 相关的文本词频数据并赋值给 temp。第 5 行代码使用 aggregate 函数做透视表，汇总统计每一个词在大类 i 相关的文本中出现的总次数，并将统计结果赋值给 temp。第 6 行代码将上一步统计的结果按照词频降序重排序。第 7 行代码使用 brewer.pal 函数在画板中取色，它的第 1 个参数表示取多少种颜色，第 2 个参数指定画板的名称。第 8 行代码启动 png 绘图设备。第 9 行代码指定画板的背景色。第 10 行代码绘制词云，words 参数时指定词汇，freq 参数指定对应的词频，scale 参数指定词显示的大小范围。这是一个标尺参数，min.freq 指定显示词频的最小值，也就是说，词频低于参数 min.freq 设置的值时，该词不会显示在词云图上，max.words 设置词云显示的最大词数，colors 指定词的颜色，random.order 制定词云中词的位置是否随机摆放，random.color 设置颜色是否随机分配，rot.per 设置词的旋转角度，font 设置字体大小的参考值，函数会根据参考值大小和标尺调整词汇显示的大小，family 设置词汇字体类型。第 14 行代码用于关闭绘图设备。第 15 行代码使用 lapply 函数将四个大类的编号赋值给匿名函数，匿名函数调用 ciyun 函数，依次批量绘制 4 张词云图。

从词云图中可以更加直接地感受到文本集中每一个阶段的内容截然不同。

## 第 12 章

# 排名就是简单的推荐系统吗？

谈到数据挖掘，自然回避不了综合排名和推荐系统。如果将推荐系统看作为用户推荐最优的信息，那么我们所做的绝大多数工作的本质都是一个推荐系统，只不过有的具体的推荐结果只是一个结论。我们比较事物的优劣、对上市公司进行估值、分析营销策略、比较新闻热度，最终都是为了给客户提供最优的决策信息。在这个过程中，最重要的能力无外乎创造概念、制定评价指标了，因为很多时候，现有的指标很难与问题一一对应。

### 12.1 全球宜居城市综合实力排行

《经济学人》基于 30 个变量对全球 140 个城市进行了综合排名，这 30 个变量共关注 5 个领域：政局稳定、基础设施、教育、医疗卫生和环境（人文和自然）。

毫无疑问，《经济学人》的全球宜居城市排名会给很多国家带上荣耀的光环，比如在排名前 5 位的城市中，加拿大就占据了 3 个，分别是温哥华、多伦多和卡尔加里。不可否认，排名总是能够引起公众巨大的兴趣，但个别专家认为，仅通过几个常规变量来衡量一个城市的宜居程度正如管中窥豹。纵观《经济学人》使用的衡量标准，却没有一条和当地居民的感受相关。

墨尔本连续 5 年位居《经济学人》的全球宜居城市排名榜榜首，之所以加拿大、澳大利亚的城市更容易登上《经济学人》的“封神榜”，主要原因是在《经济学人》采用的衡量标准中，诸如教育、医疗、基础设施等基本上属于政府行为，而加拿大、澳大利亚的地方政府在这些方面的投入非常慷慨。

人类的幸福感应该是最具有主观色彩的领域，自古人们对生活质量的理理解就莫衷一是，众说纷纭。我们没有必要就这个问题进行更加深入的讨论，这样往往会让会我们深陷泥沼，无法自拔。这里仅仅讨论的是排名方法的合理性。

这个号称基于 30 个变量，涵盖生活的 5 个主要领域的排名，真的是媒体所说的“数据驱动”的排名方式吗？媒体为其冠以“数据驱动”之名，无非为了体现排名方式的公平和公正。

首先，从算法和统计方式上来说，排名本就是一厢情愿的，无法分出优劣。为了实现什么样的

目的就选择相应的排名方法这实在谈不上公平和公正，除非大家的目的一致。排名在本质上是一个推荐系统，而推荐系统的优劣只能用推荐的东西是否符合被推荐对象的需求来评价，所以，在不同需求下比较两个推荐系统没有什么意义。

12.1.1 综合实力排行：专家法 VS 数据驱动法

排名方法一般可以被划分为两种：单变量排名和综合排名。单变量排名因为简单明了，很少会引起非议；综合排名就是使用多个变量并给予一个综合得分进行排名，这时就存在一个变量权重问题：在排名中，谁应该占据更多的权重？这简直就是谜一样的问题。

在综合排名中，确定权重有两种方法，第一种为专家法，即由该领域内的精英专家确定各个变量的权重；另外一种为数据驱动法，如使用降维技术、信息熵值等方法给各变量确定权重。很明显，专家法更加主观，而《经济学人》采取的就是专家法。下面做一个试验，假设要为政局稳定、医疗卫生、环境（文化与自然）、教育、基础设施这 5 个领域分配权重，你会怎么分配？

相关专家给这 5 个领域分配的权重，如图 12-1 所示。

分类	权重
政局稳定	25%
医疗卫生	20%
环境（人文和自然）	25%
教育	10%
基础设施	20%

图 12-1

可能很少有人和专家给出的权重不谋而合。针对专家法，即非著名的层次分析(Analytic Hierarchy Process)，R 也有专门的包与之对应，比如 pmr 包和 ahp 包都能按照专家分配的权重进行排名，具体的细节这里暂且不讲了。

由于排名数据要求内在的同质性，在理论上，无论采取哪种方法进行排名，排名的顺序应该都不会发生太大的变化。比较两个排名的相似性，可以采取等级相关检验或配对检验的方法。由于我们无法获得《经济学人》使用的详细数据，下面以另一个智库组织（多为获得诺贝尔奖的经济学家）发布的全球各国幸福指数(HPI)排名做一次试验。其使用的变量包括：生态足迹(Ecological footprint)、生活满意度(Life satisfaction)、预期寿命(Life expectancy)。下面用 3 个变量对全球 151 个国家进行综合排名。

综合排名的难点在于确定变量的权重，数据驱动法的变量权重的衡量指标当然从最简单的统计知识出发设计，比如变量解释的方差、信息熵等。这里采用主成分降维技术来确定生态足迹、生活满意度、预期寿命 3 个变量在综合排名中的权重。

• 数据整理

```
1 hpi <- read.csv(file="H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 12 章排名就是简
```

```
单的推荐系统吗/data/hpi.csv", header = T, sep = ",", row.names = 2)
2 data <- dplyr::select(hpi, -Country, -HappyPlanetIndex)
```

在做主成分分析时，基于我们的目的，这里没有必要使用主成分分析的降维功能，因为降维功能就是将多个变量汇总成比较少的主要变量，是一个从多到少的汇总过程，这中间肯定会损失一些信息。而我们要进行的是综合排名，因为算法简单，所以没必要进行降维。在上面的代码中，第 1 行代码读取数据是特别将数据的第 2 列作为行名称，一定要记住在做主成分分析、测距离、聚类时，一定要将观测值名称命名为行名称，不需要在表中保存一列用于表示观测值 case 编号，否则会报错。

- 主成分分析方差提取

```
1 datascale <- scale(data)
2 library(psych)
3 rc <- principal(datascale, nfactors = length(datascale[,1]), rotate =
"varimax", scores = TRUE)
4 loadings <- as.data.frame(unclass(rc$loadings))
#           PC2      PC1      PC3
# LifeExpectancy 0.2861240 0.8970833 0.3367116
# Wellbeing      0.3264062 0.3541963 0.8763584
# Footprint      0.9130681 0.2764263 0.2998250
5 comp <- as.data.frame(unclass(rc$scores))
6 p <- print(rc)
7 pcweight <- data.frame(p$Vaccounted)
```

在做主成分分析之前首先要进行标准化。在上面的代码中，第 1 行代码完成数据的标准化。第 2 行代码加载主成分分析包 psych。第 3 行代码进行主成分分析，其中第 1 个参数指定数据，nfactors 函数指定保留多少个主成分，这里没必要降维，所以数据有多少变量就保存多少个主成分。rotate 函数指定进行因子旋转的方法，目的是为了让主成分看起来更容易解释。当然如果不需要使用主成分解释结果，也可以不旋转。第 4 行代码加载主成分中各变量的系数（特征值），可以发现，经过方差最大正交旋转之后，第一主成分主要与 LifeExpectancy 相关，第二主成分主要与 Footprint 相关，第三主成分主要与 Wellbeing 相关。第 5 行代码提取每个观测值（case）在各个主成分上的得分，备用。第 6 行代码打印主成分的结果，并赋值给 p。第 7 行代码提取 p 中 Vaccounted 元素，转化为数据框之后存储为权重。

我们计算综合得分是通过观测值（case）在每个主成分上的得分分别乘以主成分解释的方差（Proportion Var）然后求和得来的。

- 主成分加权法综合排名

```
1 temp <- comp[, "PC1"] * pcweight["Proportion Var", "PC1"] + comp[, "PC2"]
* pcweight["Proportion Var", "PC2"] + comp[, "PC3"] * pcweight["Proportion Var",
"PC3"]
2 result <- t(as.matrix(comp)) * unlist(pcweight["Proportion Var",])
3 result <- t(result)
4 result <- apply(result, 1, sum)
5 result <- (result - min(result))/(max(result) - min(result))
```

```

6 result <- result*100
7 princompresult <- cbind(hpi, comp, result)
8 head(princompresult[with(princompresult, order(-result)), ], 10)
#
# Country LifeExpectancy Wellbeing
# 卢森堡 Luxembourg 80.0 7.1
# 卡塔尔 Qatar 78.4 6.6
# 丹麦 Denmark 78.8 7.8
# 阿拉伯联合酋长国 United Arab Emirates 76.5 7.2
# 科威特 Kuwait 74.6 6.6
# 加拿大 Canada 81.0 7.7
# 澳大利亚 Australia 81.9 7.4
# 荷兰 Netherlands 80.7 7.5
# 美国 United States of America 78.5 7.2
# 比利时 Belgium 80.0 6.9
#
# Footprint HappyPlanetIndex PC2 PC1
# 卢森堡 10.7 29.0 3.817360 -0.17747948
# 卡塔尔 11.7 25.2 4.608952 -0.34331715
# 丹麦 8.3 36.6 2.200570 -0.29886035
# 阿拉伯联合酋长国 8.9 31.8 2.799768 -0.47471839
# 科威特 9.7 27.1 3.508472 -0.62010350
# 加拿大 6.4 43.6 1.027188 0.28688826
# 澳大利亚 6.7 42.0 1.271302 0.48537410
# 荷兰 6.3 43.1 1.033790 0.33158839
# 美国 7.2 37.3 1.722807 0.02072038
# 比利时 7.1 37.1 1.710016 0.35386076
#
# PC3 result
# 卢森堡 0.3096332 100.00000
# 卡塔尔 -0.4057539 99.24288
# 丹麦 1.6434872 93.11403
# 阿拉伯联合酋长国 0.9062932 89.08164
# 科威特 0.1159964 86.35733
# 加拿大 1.7462653 85.78701
# 澳大利亚 1.2825741 85.77177
# 荷兰 1.5307082 83.52191
# 美国 1.1071746 83.23378
# 比利时 0.6847465 81.96889

```

在上面的代码中，第 1 行代码使用 `comp` 的每个 case 的主成分得分乘以相应的主成分权重，然后把它们相加，就得到了每个国家的综合得分。但是这种方法看上去比较笨拙，首先，如果有很多个主成分，则每一个都需要这样连加下去。其次，有时候自动化脚本随着数据的变化也许不知道有多少个主成分，所以要找一个替换方法。第 2~4 行代码就是替换方法，即使用矩阵和向量相乘的方法避免了按变量相乘。第 2 行代码首先将 `comp` 转化为矩阵之后再转置，然后乘以解散后的权重。第 3 行代码再将结果转置回来。第 4 行代码对矩阵的每一行求和，这里使用的是 `apply` 函数，也可以使

用 `rowsum` 函数，`temp` 和 `result` 的计算结果是一样的。第 5 行代码对结果进行归一化。第 6 行代码将结果乘以 100 改为百分制。第 7 行代码将结果和原始数据、主成分得分捆绑为一个数据框。第 8 行代码查看综合得分的前 10 名，可以看到 `result` 和 `HappyPlanetIndex` 还是存在不少的差异的。

### 12.1.2 怎么比较两个排名结果

在 12.1.1 节中我们得到了综合排名和专家排名的结果，那么怎么比较两个结果的差异？对于这种排名结果的检验，我们首先想到的可能是 `spearman` 秩相关检验。

- 排名结果比较

```
1 cor.test(x = princompresult$HappyPlanetIndex, y = princompresult$result,
method = "spearman")
# Spearman's rank correlation rho
#
# data: princompresult$HappyPlanetIndex and princompresult$result
# S = 404500, p-value = 0.0002353
# alternative hypothesis: true rho is not equal to 0
# sample estimates:
#      rho
# 0.2950513
2 plot(HappyPlanetIndex ~ result,
3      data = princompresult,
4      pch=16)
```

在上面的代码中，第 1 行代码使用 `spearman` 秩相关检验检验专家打分和主成分打分之间的相关性，原始假设认为它们之间不具有相关性，我们看到，`pvalue` 值远远小于 0.01，可以认为专家打分和主成分打分存在极显著的相关性，但是 `rho` 分比较低。从结果中也可以看出相关系数不理想，即使专家说得头头是道，但是数据却很诚实。

- 排名结果比较

```
cor.test(x = princompresult$HappyPlanetIndex, y = princompresult$result,
method = "pearson")
```

既然是得分，也可以把它们作为连续变量来检验它们之间的相关性，即可以使用 `Pearson` 系数检验，从而发现相关关系比较显著，但是相关系数比较低。

但是上面的数据毕竟都是同一个数据集通过不同统计方法上的检验结果，除了从统计学上检验结果的差异，我们经常需要引入外部数据进行验证。而衡量一个国家的幸福指数，恐怕没有比生活在其中的人们更有发言权了，经济合作与发展组织（OECD）对所属国家进行了一次群众调研，让群众给国家的表现打分，然后将得分平均综合为一个 OECD 生活指数。下面就用专家法和主成分法与 OECD 的生活指数对比一下。

- 外部数据检验

```
1 oecd <- read.csv("H:/探寻数据背后的逻辑 R 语言数据挖掘之道/第 12 章排名就是简单的
推荐系统吗/data/oecd.csv", header = T, sep = ",", stringsAsFactors = F)
```

```

2 princompresult$Chinese <- row.names(princompresult)
3 oecd <- plyr::join(oecd, princompresult[, c("Chinese", "result",
"HappyPlanetIndex")])
4 cor.test(x = oecd$HappyPlanetIndex, y = oecd$Lifesatisfaction, method =
"spearman")
5 # Spearman's rank correlation rho
6 #
7 # data: oecd$HappyPlanetIndex and oecd$Lifesatisfaction
8 # S = 5562.2, p-value = 0.09305
9 # alternative hypothesis: true rho is not equal to 0
10 # sample estimates:
11 #      rho
12 # 0.2841482
13 cor.test(x = oecd$result, y = oecd$Lifesatisfaction, method = "spearman")
14 #      Spearman's rank correlation rho
15 #
16 # data: oecd$result and oecd$Lifesatisfaction
17 # S = 1807, p-value = 4.742e-08
18 # alternative hypothesis: true rho is not equal to 0
19 # sample estimates:
20 #      rho
21 # 0.7674368

```

在上面的代码中，第 1 行代码读取 oecd 数据。第 2 行代码将 princompresult 的行名称转化为新的数据列，取名为 Chinese。通过 Chinese 将 OECD 和 princompresult 左关联。因为 OECD 调研的国家数比 princompresult 少（因为成本问题），所以进行左关联给 OECD 里的国家关联上"result"和"HappyPlanetIndex"即可。第 4 行代码使用 Spearman 检验 Lifesatisfaction 与 HappyPlanetIndex 的相关性。第 5 行代码使用 Spearman 检验 Lifesatisfaction 与 result 的相关性。

经过检验，HPI 和 OECD 的相关系数仅为 0.28，而数据驱动法的主成分加权排名与 OECD 的相关系数为 0.76，可见群众的感受和数据得分更加接近，与专家的感受相差甚远。

当然，数据驱动方法也有其缺陷，因为数据驱动法的基本思想是将各对象区分开，并非分出孰优孰劣，所以数据驱动法在方向上即和排名不相符。因为排名是要分出孰优孰劣的，但在变量变化方向比较相似时，它们又会在一定程度上“不谋而合”，这时数据驱动组织的“公平性”要远远优于专家法。

另外，有关排名，这里给读者推荐阅读一篇综述性的文章 *An R package for analyzing and modeling ranking data*。

## 12.2 协同过滤推荐系统

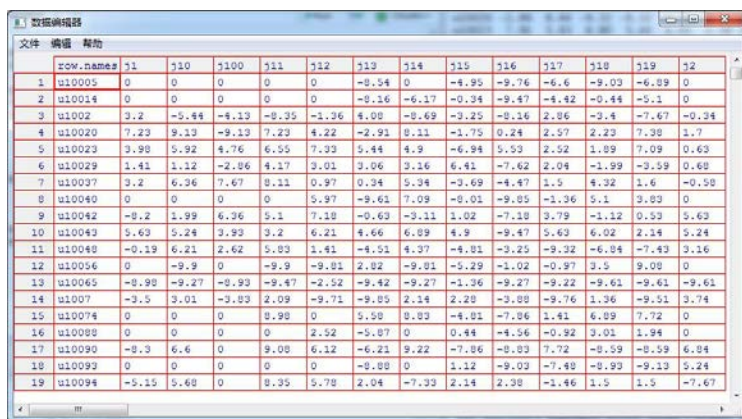
前面曾说过推荐系统几乎可以囊括所有的数据挖掘领域，比如使用 12.1 节介绍的排名系统，就可以给用户推荐宜居城市，还可以给用户推荐医院、学校、专家等这些靠综合实力排名的项目，这些都是基于内容的推荐。但是这里要讲另外一个狭义上的推荐系统：协同过滤推荐系统。所谓“协同”，可以简单理解为具有共同性，比如你要看电影，但是不知道看什么，于是你就想找口味相同的好友问问，这就是协同。

协同过滤推荐系统分为两种：基于商品的协同过滤系统（Item Based Collaborative Filtering Recommender Systems）和基于用户的协同系统过滤系统（User Based Collaborative filtering Recommender Systems）。比如一个用户选择了产品 A，我们找出和产品 A 一样被用户同时喜欢的产品，即可理解为基于商品的协同过滤系统，而基于用户的系统过滤系统则是找出和用户购买习惯相似的客户 B，并将客户 B 购买的商品推荐给客户 A。

这里无意于严格区分各种方法的界限，但是随着实际问题的复杂化，往往各种推荐系统要联合使用，所以，严格区分它们除了会引起一番论战，并没有什么非常实际的作用。

UserCF 用于反映用户的兴趣爱好，效果更好，但是计算量非常大；ItemCF 用于反映某个群体内物品使用的相似度，其结构简单，计算效率高，应该选为大规模用户推荐系统的基础框架，当然具体情况还需要具体分析。

如果从数据的角度来看，我们就很容易区分 UserCF 和 ItemCF 了，也就是对列做交叉表还是对行做交叉表的问题，这一点是区分它们的关键。来自 recommenderlab 的数据集 Jester5k 是一个包含了 5000 个匿名用户对 100 个笑话评分的数据集，几乎每个用户都对 30 个以上的笑话进行了评分（分值为[-10,10]）。下面基于这个数据分别演示 ItemCF 和 UserCF 的工作原理，但是请读者一定要记住，无论是 ItemCF 还是 UserCF，它们都是通过同一张 user（商品、电影等）× item（客户其他观测值）表计算出来的，表的行代表 user，列代表 item，如图 12-2 所示。



	row.names	1	2	3	4	5	6	7	8	9	10	11	12
1	u10005	0	0	0	0	0	-0.54	0	-4.95	-9.76	-6.6	-9.03	-6.89
2	u10014	0	0	0	0	0	-0.16	-6.17	-0.34	-9.47	-4.42	-0.44	-5.1
3	u1002	3.2	-5.44	-4.13	-0.35	-1.36	4.08	-0.69	-3.25	-0.16	2.86	-3.4	-7.67
4	u10020	7.23	9.13	-9.13	7.23	4.22	-2.91	8.11	-1.75	0.24	2.57	2.23	7.38
5	u10023	3.98	5.92	4.76	6.55	7.33	5.44	4.9	-6.94	5.53	2.52	1.89	7.09
6	u10029	1.41	1.12	-2.86	4.17	3.01	3.06	3.16	6.41	-7.62	2.04	-1.99	-3.59
7	u10037	3.2	6.36	7.67	8.11	0.97	0.34	5.34	-3.69	-4.47	1.5	4.32	1.6
8	u10040	0	0	0	0	5.97	-9.61	7.09	-0.01	-9.85	-1.36	5.1	3.83
9	u10042	-8.2	1.99	6.36	5.1	7.18	-0.63	-3.11	1.02	-7.18	3.79	-1.12	0.53
10	u10043	5.63	5.24	3.92	3.2	6.21	4.66	6.89	4.9	-9.47	5.63	6.02	2.14
11	u10048	-0.19	6.21	2.62	5.83	1.41	-4.51	4.37	-4.81	-3.25	-9.32	-6.84	-7.43
12	u10054	0	-9.9	0	-9.9	-9.81	2.82	-9.81	-5.29	-1.02	-0.97	3.5	9.08
13	u10065	-9.98	-9.27	-8.93	-9.47	-2.52	-9.42	-9.27	-1.36	-9.27	-9.22	-9.61	-9.61
14	u1007	-3.5	3.01	-3.83	2.09	-9.71	-9.85	2.14	2.28	-3.88	-9.76	1.36	-9.51
15	u10074	0	0	0	8.98	0	5.58	8.83	-4.81	-7.86	1.41	6.89	7.72
16	u10088	0	0	0	0	2.52	-5.87	0	0.44	-4.56	-0.92	3.01	1.94
17	u10090	-8.3	6.6	0	9.08	6.12	-6.21	9.22	-7.86	-0.83	7.72	-8.59	-8.59
18	u10092	0	0	0	0	0	-8.88	0	1.12	-9.03	-7.48	-8.93	-9.13
19	u10094	-5.15	5.68	0	8.35	5.78	2.04	-7.33	2.14	2.38	-1.46	1.5	1.5

图 12-2

在进行下一步分析之前，下面有必要解释一下什么是稀疏矩阵。如果矩阵中非零元素的个数远远小于矩阵元素的总数，并且非零元素的分布没有规律，则称该矩阵为稀疏矩阵（Sparse Matrix）。



这里涉及存储问题，如果以非稀疏矩阵的方式存储稀疏矩阵，就会浪费很大的内存空间。

- 稀疏矩阵和非稀疏矩阵

```
1 m1 <- matrix(0, nrow = 1000, ncol = 1000)
2 m2 <- Matrix(0, nrow = 1000, ncol = 1000, sparse = TRUE)
3 object.size(m1)
# 8000200 bytes
4 object.size(m2)
# 5632 bytes
```

可以看到，当一个矩阵为稀疏矩阵时，以非稀疏矩阵的方式存储要比以稀疏矩阵的方式存储占用的内存空间大得多，原因是稀疏矩阵只存储非零值，不再存储零值。稀疏矩阵一般以 3 列存储数据，第 1 列是行下标，第 2 列是列下标，第 3 列是非零值，不必保存零值。但随着稀疏数的减少，稀疏矩阵的存储方式逐渐多于非稀疏矩阵。R 中很多对象是以稀疏矩阵存储的，但是此方法在三元数组法的基础上进一步压缩，使用列指针，比如用于文本分析的 tm 包和 recommenderlab 包。

- 数据整理

```
1 library(recommenderlab)
2 data(Jester5k)
3 str(Jester5k)
# Formal class 'realRatingMatrix' [package "recommenderlab"] with 2 slots
# ..@ data      :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
# .. .. ..@ i      : int [1:362106] 0 1 2 3 4 5 6 7 8 9 ...
# .. .. ..@ p      : int [1:101] 0 3314 6962 10300 13442 18440 22513 27512
32512 35685 ...
# .. .. ..@ Dim    : int [1:2] 5000 100
# .. .. ..@ Dimnames:List of 2
# .. .. .. ..$ : chr [1:5000] "u2841" "u15547" "u15221" "u15573" ...
# .. .. .. ..$ : chr [1:100] "j1" "j2" "j3" "j4" ...
# .. .. ..@ x      : num [1:362106] 7.91 -3.2 -1.7 -7.38 0.1 0.83 2.91 -2.77
-3.35 -1.99 ...
# .. .. ..@ factors : list()
# ..@ normalize: NULL
temp <- Jester5k@data
str(temp)
```

在上面的代码中使用，str 函数查看 Jester5k 的结构，其中数据存储在 data 中，单独提取 data 查看它的结构，发现它是一个稀疏矩阵，其中 i 表示行标，p 是指针 (column pointer)。比如第 2 列前面有两个值，就标记为 2，第 3 列前面有 4 个值，就标记为 4。i 和 p 共同确定三元数组中的行标和列标，而 x 存储的是具体的非零值。一句话，不管是不是稀疏矩阵，你只当正常的矩阵操作就可以了。

- data.frame VS realRatingMatrix

```
1 temp <- as(Jester5k, "data.frame")
2 temp <- as(temp, "realRatingMatrix")
```

上面这两行代码展示了使用 as 函数将 data.frame 与 realRatingMatrix 相互转换的方法，也许在以

后你进行推荐预测的时候会用到。

## 12.2.1 基于商品的协同过滤系统（ItemCF）

ItemCF 首先要基于  $\text{user} \times \text{item}$  表计算各个 item 之间的相似性，然后进行推荐，具体过程大概分为两步：

- (1) 计算任意两个 item（任意两列）之间的相关性，找出每一个 item 最相关的  $k$  个 item。
- (2) 对于每一个客户，根据客户已有的 item，推荐和已有的 item 最相关的 item。

可以看出，基于  $\text{user} \times \text{item}$  表，能够找出的 item 的相关性，也只是用户行为提供的信息，比如 item a 和 item b 比较相关，只能说客户喜欢 a 同时也喜欢 b，或者客户讨厌 a 同时也讨厌 b，但是 a 和 b 真的在特征上相似吗？那倒未必。另外，理解了这种相关性的本质，可使用的评价相关性的方法也比较宽泛了，所以，基于现在的经验，即使没有专门的做协同推荐的 R 包，大多数人也可以自己写出一个协同推荐的脚本，但是有趣的是，recommenderlab 包出现了，而且非常方便。

- 训练数据集及测试数据集

```
1 CVgroup <- function(k, datasize, seed) {
2   cvlist <- list()
3   set.seed(seed)
4   n <- rep(1:k, ceiling(datasize/k))[1:datasize]
5   temp <- sample(n, datasize)
6   x <- 1:k
7   dataseq <- 1:datasize
8   cvlist <- lapply(x, function(x) dataseq[temp==x])
9   return(cvlist)
10 }
11 k <- 10
12 datasize <- nrow(Jester5k)
13 cvlist <- CVgroup(k = k, datasize = datasize, seed = 1206)
14 testnum <- unlist(cvlist[1])
15 trainnum <- unlist(cvlist[-1])
16 test <- Jester5k[testnum, ]
17 train <- Jester5k[trainnum, ]
```

这里需要强调的是，在真实案例中，在模型之前，数据整理及探索是必需的，删除那些质量不好的 user 或者 item 是提高模型效果的最佳方法，只是这里使用了样板数据，将这个重要的步骤省略了。

- 模型构建

```
1 ibcf.model <- Recommender(data = train, method = "IBCF",
                             parameter = list(k = 30, method = "Cosine"))
2 ibcf.model.profile <- getModel(ibcf.model)
3 ibcf.model.profile$sim[1:4, 1:2]
# 4 x 2 sparse Matrix of class "dgCMatrix"
#           j1           j2
```

```
# j1 .      0.19045433
# j2 0.1904543 .
# j3 0.1907928 0.06329106
# j4 .      0.11651450
```

在 R 中构建模型非常简单，只要对参数稍加了解，就可以构建模型。在上面的代码中，使用 Recommender 函数构建模型，其中 data 指定训练数据；method 指定构建模型的方法，这里选择 IBCF；parameter 是一个参数集，k 表示对每一个 item 输出多少个与之关系最近的 item，此处的 method 指定相似性的计算方式，这里选择余弦相似性，也可以选择其他计算方式，比如 Pearson 距离等。第 2 行代码用于获得模型的一些信息，信息中包括获得的 item 相似性矩阵。第 3 行代码用于查看相似性矩阵的前 4 行及前两列，从中发现笑话 1 和笑话 2、笑话 3 关系非常近。

IBCF 模型一旦建成就不需要重复构建，不仅省时省力，而且效果不错，计算量也比较小。

在测试模型之前，有必要介绍一下 IBCF 模型是怎么预测的。图 12-3 中所示的第一张图就是通过上一步模型获得的 item 相似性矩阵 `ibcf.model.profile$sim`，第二张图就是某个新用户打过分的几个笑话（商品）比如 j4 和 j5。下面要基于这两个笑话给该用户推荐其他笑话。将每一个笑话的打分作为权重，然后和相似性矩阵相乘，获得每一个准备推荐笑话的得分。

sim			newuser		Recommender	
	j1	j2			j1	j2
j4	0.116515	0.190454	j4	9.5	$0.1165145 \times 9.5$	$0.19045433 \times 9.5$
j5	0.190793	0.063291	j5	7.4	$0.1907928 \times 7.4$	$0.06329106 \times 7.4$
					总分	
					2.51875447	2.277669979

图 12-3

#### • 预测测试数据

```
1 test <- as(test, "data.frame")
#      user item rating
# 1    u17322  j1  -0.68
# 336  u17322  j2   1.75
# 703  u17322  j3  -3.16
# 1047 u17322  j4   2.77
# 1365 u17322  j5   5.49
2 test <- as(test, "realRatingMatrix")
3 ibcf.predict <- predict(object = ibcf.model, newdata = test, n = 3)
4 tempitems <- ibcf.predict@items
5 tempitems <- tempitems[lengths(tempitems) > 0]
6 tempratings <- ibcf.predict@ratings
7 tempratings <- tempratings[lengths(tempratings) > 0]
8 predictratings <- unlist(tempratings)
9 user <- names(tempitems)
10 x <- lapply(tempitems, length)
11 user <- rep(user, x)
12 tempitems <- unlist(tempitems)
13 itemname <- ibcf.predict@itemLabels
14 item <- itemname[tempitems]
```

```

15 userrecommend <- data.frame(user, item, predictratings)
16 row.names(userrecommend) <- NULL
17 head(userrecommend)
#      user item predictratings
# 1 u10105  j19      2.654358
# 2 u10105  j86      1.802461
# 3 u10105  j18      1.782789
# 4 u10115  j92      1.744853
# 5 u10115  j82      1.713545
# 6 u10115  j93      1.677771

```

在上面的代码中，第 1 行代码将测试数据由 `realRatingMatrix` 转化为数据框，可以看到数据框中包括 3 列。也就是说，在测试时，你的数据只需要准备成这样的数据框即可，然后按照下一步骤操作。第 2 行代码将数据框转化为 `realRatingMatrix`。第 3 行代码使用 `predict` 函数完成预测，`object` 用于设定构建好的模型，`newdata` 用于设定新数据集，`n` 用于设置向每个用户推荐几个 `item`，这样就可以完成测试了。测试结果存储在 S4 类对象 `ibcf.predict` 中。`ibcf.predict` 的元素 `item` 中存储了每一个用户的 `item` 编号和给用户推荐的 `item` 编号，我们使用 `@` 将其提取出来并整理一下。有的用户是没办法向其推荐的，这样他的 `item` 编号长度就为 0。第 5 行代码将这类人删除。`ibcf.predict@ratings` 存储了每一个用户被推荐 `item` 的预测评分 `rating`。第 7 行代码删除那些没有 `item` 推荐的用户的。第 8 行将评分解散为向量备用；我们要将数据整理成两列，第 1 列是用户 `id`，第 2 列是用户备推荐的 `item`，`tempitems` 是一个 `list`，它的每一个元素的名称就是用户 `id`，每一个元素包含的向量存储了 `item` 的编号，我们只需要将用户 `id` 复制  $n$  次， $n$  等于该用户下 `item` 编号的个数即可。第 9 行代码提取每个用户的 `id`。第 10 行代码获得每个用户下的编号个数。第 11 行代码将用户 `id` 复制相应的次数。第 12 行代码将 `tempitems` 解散为向量，向量存储了 `item` 的编号，不是名称，我们需要用编号提取名称。第 13 行代码提取每个 `item` 的名称。第 14 行代码使用编号提取对应编号的 `item` 名称。第 15 行代码将用户 `id`、`item` 名称、`ratings` 捆绑为数据框，这个数据框就是新数据用户的推荐结果。第 16 行代码将行名称赋值为 `NULL`。

- 验证整理是否正确

```

1 itemnum <- ibcf.predict@items[[1]]
2 recc_predicted@itemLabels[itemnum]
# [1] "j19" "j86" "j18"
3 recc_predicted@ratings[[1]]
# [1] 2.654358 1.802461 1.782789

```

在上面的代码中，第 1 行代码提取第一个客户推荐的 `item` 编号。第 2 行代码查看相应的 `item` 名称。第 3 行代码查看 `item` 的 `ratings` 得分。

## 12.2.2 基于用户的系统过滤系统（UserCF）

基于用户的协同过滤系统（UserCF）和基于商品的协同过滤系统（ItemCF）使用相同的表，只不过它计算的是用户之间的相似度，所以具体过程至少应该分为以下两步。

(1) 计算用户相似度：计算每一个用户与新用户（不是全新用户）的相似度，方法可以选择计算相似度和余弦相似性，然后通过选取相似度排序在前  $k$  个或者相似度超过某个阈值的用户作为相似用户。

(2) 计算平均得分：计算  $k$  个相似用户的每个 item 的平均得分（rating，即  $k$  个相似用户为某个 item 打分的平均值），或者计算得分与相似度乘积的加权得分，再计算平均加权得分，然后根据平均得分或平均加权得分筛选出前  $n$  个 item 作为推荐 item。

从上面的步骤中，我们可以清楚地看到 ItemCF 与 UserCF 的区别，以及它们与基于内容的推荐系统（最优内容）的区别，将概念放在具体的实例中讨论，这才是讨论概念的正确方式。

- 训练数据集及测试数据集

```
1 testnum <- unlist(cvlist[1])
2 trainnum <- unlist(cvlist[-1])
3 test <- Jester5k[testnum, ]
4 train <- Jester5k[trainnum, ]
```

在上面的代码中使用相同的抽样结果提取训练数据集和测试数据集，作为备用数据。

- 模型构建

```
1 ubcf.model <- Recommender(data = train, method = "UBCF",
                             parameter = list(nn = 30, method = "Cosine"))
2 ubcf.model.profile <- getModel(ubcf.model)
3 ubcf.model.profile
```

在上面的代码中，第 1 行代码构建基于用户的协同过滤模型，nn 指定计算多少个相似用户，method 指定相似度的计算方法。第 2~3 行代码用于查看模型包括哪些东西，模型的 data 元素竟然包含了所有的训练数据集。这一步基本上没有构建模型，只是存储了模型的一些参数和数据，只能在预测数据时拿新用户和模型里存储的用户计算相似性，然后筛出来前 nn 个用户。

从上面可以发现，在 UBCF 的训练数据集中，如果提取所有老客户数据，则每次计算量可能非常大，解决方法是可以筛选一个非常具有代表性的客户数据集作为训练数据集即可，这样就可以大大减少计算量，而且训练数据集也不需要每天更新，这应该是一个可行的降低 UBCF 的计算量的方法。UBCF 和 IBCF 在这一点上的不同，导致前者预测过程慢，而模型构建快，而后者模型构建慢，预测则比较快。

- 预测测试数据

```
1 test <- as(test, "data.frame")
#       user item rating
# 1    u17322  j1  -0.68
# 336  u17322  j2   1.75
# 703  u17322  j3  -3.16
# 1047 u17322  j4   2.77
# 1365 u17322  j5   5.49
2 test <- as(test, "realRatingMatrix")
3 ubcf.predict <- predict(object = ubcf.model, newdata = test, n = 3)
4 tempitems <- ubcf.predict@items
5 tempitems <- tempitems[lengths(tempitems) > 0]
```

```
6 tempratings <- ubcf.predict@ratings
7 tempratings <- tempratings[lengths(tempratings) > 0]
8 predictratings <- unlist(tempratings)
9 user <- names(tempitems)
10 x <- lapply(tempitems, length)
11 user <- rep(user, x)
12 tempitems <- unlist(tempitems)
13 itemname <- ubcf.predict@itemLabels
14 item <- itemname[tempitems]
15 userrecommend <- data.frame(user, item, predictratings)
16 row.names(userrecommend) <- NULL
17 head(userrecommend)
#      user item predictratings
# 1 u10105  j36      1.577583
# 2 u10105  j35      1.374615
# 3 u10105  j49      1.143006
# 4 u10115  j72      1.630788
# 5 u10115  j81      1.607220
# 6 u10115  j76      1.499491
```

UBCF 的预测数据整理方式和 IBCF 一样，但是它们的预测结果差异很大，那么问题来了：哪个更好呢？

## 12.2.3 推荐系统效果评比

要比较，首先要构建比较的指标，这里不再使用比较连续变量的指标，仅仅比较准确率和召回率，因为我认为这里的推荐只能用正确和错误评价。一般来说，准确率（precision）表示模型检索出来的条目中有多少是正确的，召回率（recall）表示测试样本中所有正确的条目有多少被检索出来。为什么要用英文标志？因为使用中文一般容易混淆。

基于以上指标，下面用 IBCF 模型测试一下，但是既然要测试，测试数据集就要改造一下，我们给测试数据集中的每个用户删除几个 item，用 IBCF 模型基于剩余的 items 进行推荐，计算推荐结果与删除的 item 之间的差异就可以了。

- 测试数据调整

```
1 test <- as(test, "data.frame")
2 temp <- 1:length(test[,1])
3 temp <- split(temp, test$user)
4 temp <- lapply(temp, function(x) x[-sample(1:length(x), 4)])
5 temp <- unlist(temp)
6 itemrm <- test[-temp,]
7 test <- test[temp,]
8 test <- as(test, "realRatingMatrix")
```

在上面的代码中，第 1 行代码将 test 转化为数据框。第 2 行代码生成一个和 test 等长的序列，

作为提取 test 数据的行标。第 3 行代码将 temp 按照用户分组。第 4 行代码将每组用户随机删除第 4 个行标。第 5 行代码将 temp 为向量。第 6 行代码提取不在 temp 行标里的数据，即刚刚删除的 item，备用。第 7 行代码提取保留的 item 作为测试数集，将测试数集转化为 realRatingMatrix 对象就可以预测了。

- 预测测试数据

```
1 ibcf.predict <- predict(object = ibcf.model, newdata = test, n = 3)
2 tempitems <- ibcf.predict@items
3 tempitems <- tempitems[lengths(tempitems) > 0]
4 tempratings <- ibcf.predict@ratings
5 tempratings <- tempratings[lengths(tempratings) > 0]
6 predictratings <- unlist(tempratings)
7 user <- names(tempitems)
8 x <- lapply(tempitems, length)
9 user <- rep(user, x)
10 tempitems <- unlist(tempitems)
11 itemname <- ibcf.predict@itemLabels
12 item <- itemname[tempitems]
13 userrecommend <- data.frame(user, item, predictratings)
14 row.names(userrecommend) <- NULL
15 userrecommend$rectype <- rep(1, length(userrecommend[,1]))
```

测试的代码没有大的改变，只在最后一列添加了 type 标记列。

- 计算准确率

```
1 itemrm$typerm <- rep(1, length(itemrm[,1]))
2 temp <- plyr::join(userrecommend[, c("user", "item")], itemrm[, c("user",
"item", "typerm")])
3 precision <- sum(temp$typerm, na.rm = T)/length(temp[, 1])
```

在上面的代码中，第 1 行代码给 itemrm 添加新列 typerm，用于标识被移除的 item。第 2 行代码将推荐的 item 和移除的 item 按照 user、item 左联接，如果推荐正确，则 temp 的 typerm 列就为 1，否则为 NA。第 3 行代码将正确推荐的 item 个数除以推荐的 item 总个数，获得推荐的准确率，结果为 0.05 左右。

- 计算召回率

```
1 temp <- plyr::join(itemrm[, c("user", "item")], userrecommend[, c("user",
"item", "rectype")])
2 recall <- sum(temp$rectype, na.rm = T)/length(temp[, 1])
```

在上面的代码中，第 1 行代码将移除的 item 和推荐的 item 按照 user、item 左联接，如果推荐正确，则 temp 的 rectype 列为 1，否则为空。第 2 行代码将正确推荐的 item 个数除以移除的 item 总个数，获得召回率，结果为 0.0385 左右。

按照上述方法，稍加改造就可以作为推荐系统的评估方法，你可以测试 UBCF 或设计交叉检验模型比较不同推荐系统的效果。

# 第 13 章

## 生物信息学中的数据挖掘案例

### 13.1 生物信息学与 R 语言

R 语言最初的两位设计者 Ross Ihaka 和 Robert Gentleman，在 20 世纪 90 年代便成了新西兰奥克兰大学统计系的教授。其中 Robert Gentleman 一直致力于利用 R 语言在生物信息学方面的开发工作。从 2001 年起，Gentleman 与西雅图弗雷德·哈金森癌症研究中心（Fred Hutchinson Cancer Research Center, Seattle, WA）的开发团队以及其他多个国际研究机构，发起了 Bioconductor 计划，旨在推动发展生物信息学和计算生物学的开源软件。2015 年 4 月，Gentleman 出任个人基因组和生物技术公司 23andMe 的副总裁，继续在该领域发挥他在生物信息学和计算药物发掘方面的专长。

### 13.2 生物信息学中常用的软件包

#### 13.2.1 软件包简介

R 语言在最近几年受到越来越广泛的欢迎，这和各个领域中 R 包（R Package，或称 R 软件）不断增加密切相关。一个常见的标准化 R 包里含相关的 R 代码、数据集、使用文档以及测试命令。我们需要把 R 包安装到 R 环境中后才能使用其中的命令或数据。当然，一些基本的 R 包，如 base，是在安装 R 软件的时候就已经被默认安装到 R 环境中了，但更多的 R 包，如大名鼎鼎的 ggplot2 包，则需要自行下载并安装才能使用。

大部分（当然不会是全部）的 R 包都可以在 CRAN 上找到。这里简单介绍一下 CRAN，即 Comprehensive R Archive Network（R 综合文档网络），是存放 R 软件、R 包及文档等资源的公开镜



像群。目前国内已有清华大学、北京大学、厦门大学、中国科学院等几个单位设立公开镜像提供下载服务。在 R 环境下使用命令：

```
1 install.packages(#R 包#)
```

即可安全快捷地下载安装需要的 R 包。也可以直接登录 CRAN 镜像网站，在网站左侧栏目中有一个 Task Views 的超链接，能根据任务需求来寻找相应的 R 包。

除了 CRAN，很多专业领域都提出了 R 语言的具体应用计划。前面提到的 Bioconductor 计划，就提供了众多高通量基因组测序数据（high-throughput genomic data）分析工具。另外，还有很多做生物信息的专家会在 GitHub 上发布 R 包。我们可以在 Rdocumentation 或者 inside-R 这类网站中搜索需要的 R 包。

这些众多的 R 包由全世界的作者所贡献，所以难免会发生一些不兼容的情况，这时候可以利用 Packrat 或者 CheckPoint 来检查及管理 R 包的安装、卸载及更新。Windows 系统下的 R 用户也可以用 installr 包的 updateR 函数来更新 R 软件的版本。

## 13.2.2 数据表示方式——对象类（class）

在使用 R 语言进行数据分析之前，先要理解一个关键问题——数据的类型，即数据的表示方式（对象类，class）。就像数据有本身自有的数据结构（或者叫文件格式）一样，例如我们常说的 MP3 格式、PDF 格式或者 FASTA 格式，在 R 中，每个对象都有对象类的属性，这个属性定义了这个对象中数据的格式。

查看一个对象的属性，就像在 Window 环境下，要了解一个文件的文件格式一样简单和必需，具体如下代码所示。

```
1 x=10
2 class(x)
# [1] "numeric"
3 class(class(x))
# [1] "character" #把命令 class(x)的输出结果看成一个对象，其属性则是字符
```

在生物信息学领域，需要处理的数据种类很多，我们必须时刻关注这些信息数据的格式是什么，大致的数据结构是怎样的。只有知道每个对象的类别，才能知道该用哪些 R 包和函数工具来进行数据分析，挖出我们想要的“金子”。

## 13.2.3 生物信息学 R 包简介：Bioconductor 和 CRAN

### 1. CRAN

CRAN，即 Comprehensive R Archive Network（R 综合典藏网），我们除了可以在这里下载各种 R 软件版本的源代码，还可以下载各种牛人撰写并共享的 R 包。这些 R 包按照任务类别被存放在 CRAN Task Views 栏目里，其中与生物相关的有 Genetics（遗传学）和 Phylogenetics（系统发育学）这两个

栏目。

这两个栏目里的 R 包都是经过专家筛选的。Genetics 栏目的维护员（Maintainer）是英国伦敦帝国学院的 Giovanni Montana 教授，他在栏目里介绍了在处理群体遗传学研究数据时可能会用到的统计方法及算法。Phylogenetics 栏目则由美国田纳西大学的副教授 Brian O'Meara 维护，其中主要介绍了系统发育学和群体遗传学研究中常见的 R 包。本章会主要介绍分子系统发育学在 R 平台上的应用基础，其他如芯片表达数据、RNA-seq 等与基因表达相关的 R 包，在很多书籍里都有详细介绍，这里就不介绍了，如表 13-1 所示。

表 13-1

R 包	功能简介
ape	处理序列数据，对系统发育树进行读写、绘图和加工，进行祖先状态分析、多样化和宏进化的分析，计算等位基因在核苷酸水平的距离，进行 Mantel 检验等。CRAN 上有超过 100 个 R 包依赖 ape
phyclust	利用系统发育的方法对 DNA 序列进行聚类分析，并附带了常用序列生成软件 ms 和 seq-gen
phytools	功能强大的系统发育分析包
phangorn	构建系统发育树，支持最大似然法和最大简约法
genetics	能处理多位点的基因型和单倍型数据，可以计算位点频率、检验 Hardy-Weinberg 平衡和遗传连锁等
adegenet 和 pegas	可以用于分析微卫星数据，也可以用于分析 SNP 数据；同时可以进行多维分析（DAPC, sPCA）以及作图、统计检验、数据模拟、遗传距离计算等
PopGenome	针对群体基因组数据分析工具，能直接处理 gff、vcf 等格式的高通量数据，同时整合了很多有用的群体统计函数

2. Bioconductor

Bioconductor 是一个基于 R 语言开发的处理高通量基因组数据的开源软件工具集。该工具集的开发计划始于 2001 年，最初由美国的西雅图弗雷德·哈金森癌症研究中心的一群做数据分析的生物学家发起，逐渐地，该计划吸引了全美国乃至全世界范围的学者参与进来。除了每年更新两次 R 软件工具集，Bioconductor 还共享了大量的生物学实验数据，涵盖了通路分析、生物芯片和其他注释数据。

13.2.4 ape 包

ape 包是 R 语言在生物信息学领域的基础 R 包，主要应用在系统发育学上，同时可以对基因序列或者蛋白序列数据进行读写、操作和加工，并在序列数据的基础上进行多种运算及检验。

无论是 CRAN 还是 Bioconductor 上的 R 包，所有它们自带的说明书都编写得有点反人类——命令介绍全部按照字母顺序排列，这很难说得上有逻辑。这对于急着要大干一场的初学者十分不利。

因此建议读者可以从后面的索引读起，起码那里会把命令按照主题分成几类（但有点儿让人崩溃的是，这些主题也是按照字母顺序排列的）。ape 包的说明书也不例外，不过不用担心，下面会重新根据功能和内容对 ape 包的相关函数和数据进行分类介绍，并且对一些生物信息学中常见的数据进行简单介绍。

## 1. 数据读取、清洗、检查

安装一个 R 包后，我们可以用几个简单的命令了解里面装了什么东西，其中 lsf.str 函数可以获得 R 包里的函数列表，data 函数可以获得 R 包自带的数据样本。

- 样本数据

```
1 install.packages("ape") ### 没安装 ape 的记得要先装这个 R 包
2 library("ape")
3 lsf.str("package:ape")
#$.multiPhylo : function (x, name)
#$<-.multiPhylo : function (x, ..., value)
# [.AAbin : function (x, i, j, drop = FALSE)
# [.DNABin : function (x, i, j, drop = FALSE)
# [.multiPhylo : function (x, i)
#.....
4 data(package="ape")$results
#   Package LibPath                               Item
# [1,] "ape"      "d:/Program Files/R/R-3.3.2/library" "HP.links"
# [2,] "ape"      "d:/Program Files/R/R-3.3.2/library" "bird.families"
# [3,] "ape"      "d:/Program Files/R/R-3.3.2/library" "bird.orders"
# [4,] "ape"      "d:/Program Files/R/R-3.3.2/library" "carnivora"
# [5,] "ape"      "d:/Program Files/R/R-3.3.2/library" "chiroptera"
# [6,] "ape"      "d:/Program Files/R/R-3.3.2/library" "cynipids"
# [7,] "ape"      "d:/Program Files/R/R-3.3.2/library" "gopher.D"
# [8,] "ape"      "d:/Program Files/R/R-3.3.2/library" "hivtree.newick"
# [9,] "ape"      "d:/Program Files/R/R-3.3.2/library" "hivtree.table"
#[10,] "ape"      "d:/Program Files/R/R-3.3.2/library" "landplants.newick"
#[11,] "ape"      "d:/Program Files/R/R-3.3.2/library" "lice.D"
#[12,] "ape"      "d:/Program Files/R/R-3.3.2/library" "lmorigin.ex1"
#[13,] "ape"      "d:/Program Files/R/R-3.3.2/library" "lmorigin.ex2"
#[14,] "ape"      "d:/Program Files/R/R-3.3.2/library" "mat3"
#[15,] "ape"      "d:/Program Files/R/R-3.3.2/library" "mat5M3ID"
#[16,] "ape"      "d:/Program Files/R/R-3.3.2/library" "mat5Mrand"
#[17,] "ape"      "d:/Program Files/R/R-3.3.2/library" "opsin.newick"
#[18,] "ape"      "d:/Program Files/R/R-3.3.2/library" "woodmouse"
#   Title
# [1,] "Test of host-parasite coevolution"
# [2,] "Phylogeny of the Families of Birds From Sibley and Ahlquist"
# [3,] "Phylogeny of the Orders of Birds From Sibley and Ahlquist"
```

```
# [4,] "Carnivora body sizes and life history traits"
# [5,] "Bat Phylogeny"
# [6,] "NEXUS Data Example"
# [7,] "Test of host-parasite coevolution"
# [8,] "Phylogenetic Tree of 193 HIV-1 Sequences"
# [9,] "Phylogenetic Tree of 193 HIV-1 Sequences"
#[10,] "Gene Tree of 36 Landplant rbcL Sequences"
#[11,] "Test of host-parasite coevolution"
#[12,] "Multiple regression through the origin"
#[13,] "Multiple regression through the origin"
#[14,] "Three Matrices"
#[15,] "Five Trees"
#[16,] "Five Independent Trees"
#[17,] "Gene Tree of 32 opsin Sequences"
#[18,] "Cytochrome b Gene Sequences of Woodmice"
```

ape 包中提供的样本数据均来源于已公开发表的研究数据，其中包括蛋白/基因序列、系统发育树，以及生物学形态描述。可以用 `data(#数据#)` 命令直接加载这些数据。表 13-2 中列出了 ape 包中常见的样本数据。读者可以利用这些数据来练习。

表 13-2 ape 包中常用的样本数据

数 据	内 容	类 (class)
cynipids	8 种瘿蝇的蛋白序列	List [NEXUS 格式]
woodmouse	15 个小林姬鼠的 Cytochrome b 基因序列	DNABin
hivtree.newick	HIV-1 病毒 M 组序列构建的系统发育树	character [newick 格式]
opsin.newick	32 个视蛋白基因序列构建的系统发育树	character [newick 格式]
landplants.newick	36 种陆生植物 rbcL 基因构建的系统发育树	character [newick 格式]
bird.families	鸟类系统发育树 (科)	phylo
bird.orders	鸟类系统发育树 (目)	phylo
chiroptera	蝙蝠系统发育树	phylo
carnivora	112 种食肉目动物体型及特征	data.frame
mat3	27 个观察对象的 9 个变量	data.frame
mat5M3ID	250 个观察对象的 50 个变量	data.frame
mat5Mrand	250 个观察对象的 50 个变量	data.frame

有些 R 包可能会带有样品程序，可以用 `demo` 函数调用：

```
1 demo(ape)
# Error in demo(ape) : No demo found for topic 'ape'   ### 好尴尬，ape 软件包没有 demo。
```

除了要了解 ape 包自带的的功能，更重要的是要知道如何读写我们需要处理的生物信息学数据。一般而言，需要使用 ape 包处理的数据基本是生物序列或生物的形态特征。ape 包有多个函数可以实

现处理这些数据的功能。

`read.dna` 函数可以读取本地的 DNA 序列文件,序列文件格式可以是 `phylip`(一般文件后缀是`.phy`)、`clustal`(`.aln`)、`fasta`(`.fas`)。`fasta` 格式的文件还可以直接用 `read.FASTA` 函数读取。这里我们先“制造”一些序列格式的文件,然后利用 `read.dna` 函数读取这些文件。函数会返回一个名为 `DNABin` 的类,用来存储序列。

- 读取数据

```
1 cat(
2 " 3 15",                               #首行交代下列有 3 个序列, 每个序列 15 个碱基。
3 "Seq1      ATCGATCGTACTAAA",          #第 2~4 行为每个序列的名字和 DNA 序列。
4 "Seq2      ATCGATCGTACTAAA",
5 "Seq3      ATCGATCGTACTAAA",
6 file = "exdna1.phy", sep = "\n")
7 read.dna("exdna1.phy", format = "sequential", as.character=TRUE) #读取生
成的 phy 文件, 以字符串的形式读取以进行检查。
8 ex.dna1 <- read.dna("exdna1.phy", format = "sequential") #读取 phy 文件,
以 DNABin 类形式存储。
9 str(ex.dna1)
10 ex.dna1
```

生成一小段 DNA 序列,保存为 `phy` 格式的序列文件,`read.dna` 同样支持另一种 `phylip` 格式。

- 读取数据

```
1 cat(
2 " 3 40",
3 "Seq1      ATCGATCGTA CTAAAATCGA",
4 "Seq1      ATCGATCGTA CTAAAATCGA ",
5 "Seq2      ATCGATCGTA CTAAAATCGA ",
6 "          TCGTACTAAA TCGTACTAAA",
7 "          TCGTACTAAA TCGTACTAAA",
8 "          TCGTACTAAA TCGTACTAAA",
9 file = "exdna2.phy", sep = "\n")
10 ex.dna2 <- read.dna("exdna2.phy")
```

- `clustal` 输出的结果

```
1 cat(
2 "CLUSTAL (ape) multiple sequence alignment", "",
3 "Seq1      NTTGACTACTAAAANTTATCAGTCACT",
4 "Seq2      ATTCGACTACTAAAAATTATCAACCACT",
5 "Seq3      ATTCGACTACTAAAAATTATCAATCACT",
6 "          *****",
7 file = "exdna3.aln", sep = "\n")
8 ex.dna3 <- read.dna("exdna3.aln", format = "clustal")
```

- FASTA 格式

```
1 cat(
```

```

2 ">Seq1",
3 "NTTCGACTACTAAAAATTATCAGTCACT",
4 ">Seq2",
5 "ATTCGACTACTAAAAATTATCAACCACT",
6 ">Seq3",
7 "ATTCGACTACTAAAAATTATCAATCACT",
8 file = "exdna4.fas", sep = "\n")
9 ex.dna4 <- read.dna("exdna4.fas", format = "fasta")
10 ex.dna4 <- read.FASTA("exdna4.fas") #也可以直接用函数 read.FASTA 读取 fasta
文件。

```

read.GenBank 函数可以通过基因登记号（accession id）下载 GenBank 数据库中的序列。这里可以用 read.GenBank 函数下载所有小林姬鼠（woodmouse）的 Cytochrome b 基因序列（Michaux et al. 2003）。

```

1 ref = paste("AJ",c(seq(511877,511887), seq(511889,511893), seq(511896,
511926), seq(511928, 511932), seq(511935, 511986))), sep="")
2 woodmouse_all = read.GenBank(ref)

```

- 为何要排除 AJ511888、AJ511894、AJ511895、AJ511927、AJ511933、AJ511934 这 6 个基因记号？本章 13.2.6 节会有介绍。

```

1 woodmouse_all
# 104 DNA sequences in binary format stored in a list.
# Mean sequence length: 878.221
#   Shortest sequence: 157
#   Longest sequence: 965
# Labels: AJ511877 AJ511878 AJ511879 AJ511880 AJ511881 AJ511882 ...
# Base composition:
#   a   c   g   t
# 0.308 0.259 0.125 0.308

```

如果序列是原始序列，譬如从网上的数据库下载的，或者是测序公司给你的第一手数据，那么这时候最好先检查一下序列是否比对好了。上面的信息提示了 104 条序列平均长度、最短长度及最长长度，这说明这些序列还没有进行比对。只有比对好的序列才能进行后续的数据分析。做序列比对工作的软件有很多，Clustal、Muscle、SeqMan、Mega 都可以。ape 包里面也有对应的函数可以调用这些命令来进行序列比对：clustal()、muscle()和 tcoffee()。以 clustalw 为例，只要把该软件路径指定到环境变量中，即可顺利使用。该软件的下载路径是 <http://www.clustal.org/download/current/>。

```

1 woodmouse_all_aln = clustal(woodmouse_all, exec="clustalw2") #参数 exec
用于指定 clustalw2 程序的调用方法。当然，我们需要耐心等待 104 条 DNA 序列两两比对完成。
2 woodmouse_all_aln
# 104 DNA sequences in binary format stored in a matrix.
# All sequences of same length: 965
# Labels: AJ511930 AJ511982 AJ511956 AJ511957 AJ511923 AJ511950 ...
# Base composition:
#   a   c   g   t

```

```
# 0.308 0.259 0.125 0.308
```

比对完成后，系统会提示所有序列的长度都是 965 了，至于短的序列为什么在比对后变长了，其实只是往序列里加入了“-”（gap）而已。到这里，我们已经成功把生物学研究中最基本的一类数据——DNA 序列（这里是 DNABin）成功读入并且可以用于后续分析了。

拿到这些序列后，我们最好还是先看一下数据质量如何。在完成序列比对后，可以用 `as.character.DNABin()` 命令来检查序列；当然，更好的办法是用 `image.DNABin()` 命令检查。

- 数据质量检查与清洗（见图 13-1）

```
image.DNABin(woodmouse_all_aln)
```



图 13-1

也许数据太多会不容易明白，下面将一个数据小样本放大局部看看就清楚了（见图 13-2）：

```
1 data(woodmouse)
2 image.DNABin(woodmouse[1:7,1:50],legend = T)
3 s = 50; n = 7; x = c( ); y = c( )
4 for (i in 1:n){
5   y=c(y, rep(i, s))
6   x=c(x, 1:s)
7 }
8 text(x, y, toupper(t(as.character(woodmouse[1:7,1:50])))) ### 把 ATCG 也
添加上去。
```

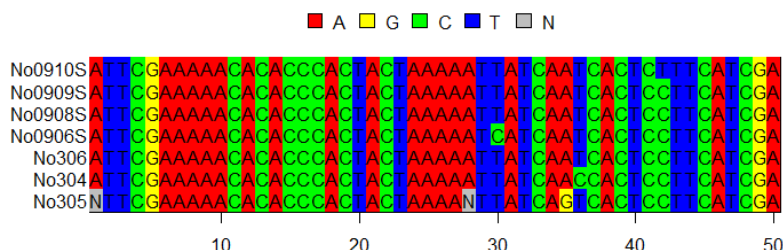


图 13-2

最新版本的 ape 包给序列作图的顺序是从上到下的，所以，要注意构建 Y 坐标轴向量的顺序（见图 13-3）：

```
1 for (i in n:1){ #将上边的 1:n 改为 n:1，则 y 坐标轴就从上到下构建了。
2   y=c(y, rep(i, s))
3   x=c(x, 1:s)
4 }
5 text(x, y, toupper(t(as.character(woodmouse[1:7,1:50]))))
```

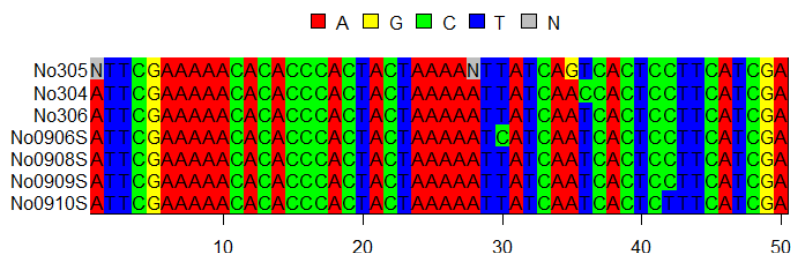


图 13-3

图 13-4 中的每一行代表一条 DNA 序列，每一列代表这个基因在每条序列上的碱基状态，这些碱基是用不同颜色表示的。因为这是同一个物种的群体基因数据，所以能看到像条形码一样的图像，这说明总体的比对质量还算不错。但是，我们还是可以看到有些序列“开天窗”，黑色占了很大比例。出现这种情况一般有两种原因：第一，序列的碱基片段缺失，表明这个生物个体携带了缺失片段的基因（尤其是当这个缺失片段特别长的时候，你可以好好准备一下，然后跟实验室老板汇报你的幸运发现）；第二，测序公司没有把这个基因的扩增产物测通（这时你该打电话给 DNA 测序公司的技术员，让他们把样品重测一遍）。显然，第二种原因大概能解释 99.9% 的情况。在重新获取新的 DNA 序列数据之前，我们也可以先挑一些能用的数据来进行后续分析。如果序列缺失太多（譬如空白数据超过 25%，这个值只是举例，具体还得看数据的质量和后续分析容忍数据缺失的程度），我们可以清洗这些序列：

```
1 gap_percent = apply(as.character(woodmouse_all_aln), 1, function(x)
{table(x)["-"]/length(x)}) #运用 apply 来迭代计算每个片段中 gap 的比例
2 gap_percent[is.na(gap_percent)]=0 #由于有些序列的没有 gap，所以这些序列会返回
```



```
NA 值，需要把这些 NA 值改为 0
3 woodmouse_analyse = woodmouse_all_aln[gap_percent<=0.25,] #保留 gap 比例
  不超过 25%的序列进行下一步分析
4 woodmouse_analyse
# 90 DNA sequences in binary format stored in a matrix.
# All sequences of same length: 965
# Labels: AJ511877 AJ511878 AJ511879 AJ511880 AJ511881 AJ511882 ...
# Base composition:
# a      c      g      t
# 0.308 0.259 0.125 0.308
5 image.DNAbin(woodmouse_analyse) #将没测好的序列被去掉后，整体数据干净多了
```



图 13-4

当序列数据准备好后，可以开动“挖掘机”在数据中寻找信息了。从信息的角度来看，DNA 仅仅是由 A、T、C、G 这 4 种字母（碱基对）组成的，但是组成的数据量却大得惊人，譬如人类基因组（DNA）大约由 30 亿个字母组成。前面提到的 woodmouse 数据集中有 15 个 DNA 序列，比对后每个 DNA 序列有 965 个碱基对。其中绝大部分碱基对在 15 个个体中都是一样的，只有很小一部分碱基对在一些个体中不一样（基因变异），而这些有差异的位点，正是需要我们挖掘出来的信息位点。下面还是以 ape 包中自带的 woodmouse 数据为例来介绍。

• 查找变异位点

```
1 seg.sites(woodmouse)
# [1] 30 33 35 36 42 51 54 60 72 96 106 123 201 213 234 237
  279 291
```

```
#[19]297    306 314 316 318 340 342 343 349 365 409 417 438 456 462 477
510 514
#[37]534    540 546 576 591 672 675 684 697 715 738 810 837 876 909 920
957 959
#[55]960    963      #共有 56 个位点，我们叫这些位点叫“信息位点”
```

把这些信息位点列出来（见图 13-5）。

```
1 image.DNAbin(woodmouse[,seg.sites(woodmouse)])
2 s = length(seg.sites(woodmouse))
3 n = nrow(woodmouse)
4 x=c(); y=c()
5 for (i in 1:n){
6   y=c(y, rep(i, s))
7   x=c(x, 1:s)
8 }
9 text(x, y, toupper(t(as.character(woodmouse[,seg.sites(woodmouse)]))))
```

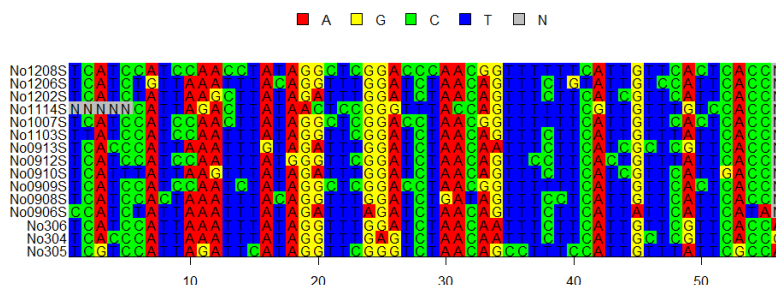


图 13-5

虽然，我们已经把所有的信息位点列了出来，但是这些序列之间的关系还是不清晰，所以接下来尝试寻找序列之间的关系。

## 2. 序列数据关系研究——构建系统发育树

ape 包的一个主要功能是构建系统发育树，利用遗传距离（譬如不同个体上同一个基因片段的 DNA 差异程度）来计算每个序列之间的亲缘距离，从而完成对生物序列数据的亲缘关系分析和可视化。

这个原理不复杂，操作起来也很简单。在获得干净序列的情况下，我们只需要经过以下两步即可。

```
1 d <- dist.dna(woodmouse)
2 tree <- nj(d) ###用 NJ 法构建系统发育树，生成一个 phylo 类
3 plot.phylo(tree, type= "unrooted") #画出构建的系统树
```

其中对象 d 的矩阵记录了两两序列的遗传距离，如图 13-6 所示。

	No305	No304	No306	No0906S	No0908S	No0909S	No0910S	No0912S	No0913S	No1103S	No1007S	No1114S	No1202S	No1206S
No304	0.0145													
No306	0.0134	0.0033												
No0906S	0.0179	0.0122	0.0089											
No0908S	0.0168	0.0111	0.0078	0.0122										
No0909S	0.0167	0.0156	0.0122	0.0167	0.0156									
No0910S	0.0167	0.0111	0.0077	0.0077	0.0111	0.0156								
No0912S	0.0145	0.0133	0.0100	0.0145	0.0133	0.0111	0.0133							
No0913S	0.0179	0.0055	0.0044	0.0111	0.0122	0.0167	0.0077	0.0145						
No1103S	0.0111	0.0100	0.0066	0.0111	0.0100	0.0078	0.0100	0.0033	0.0111					
No1007S	0.0167	0.0156	0.0122	0.0167	0.0156	0.0022	0.0156	0.0111	0.0167	0.0078				
No1114S	0.0156	0.0167	0.0156	0.0213	0.0213	0.0213	0.0201	0.0190	0.0190	0.0156	0.0190			
No1202S	0.0168	0.0111	0.0078	0.0077	0.0111	0.0156	0.0022	0.0133	0.0078	0.0100	0.0133	0.0179		
No1206S	0.0167	0.0122	0.0089	0.0111	0.0100	0.0167	0.0100	0.0145	0.0133	0.0111	0.0167	0.0224	0.0100	
No1208S	0.0190	0.0179	0.0145	0.0190	0.0179	0.0022	0.0179	0.0134	0.0190	0.0100	0.0022	0.0213	0.0156	0.0190

图 13-6

对象 tree 则是生成的系统发育树，属于 phylo 类。

```
1 tree
#Phylogenetic tree with 15 tips and 13 internal nodes.
#
#Tip labels:
#   No305, No304, No306, No0906S, No0908S, No0909S, ...
#
#Unrooted; includes branch lengths.
```

上面的最后一行代码把系统树以无根树的形式画出来，各个序列之间的亲缘关系一目了然（见图 13-6）。

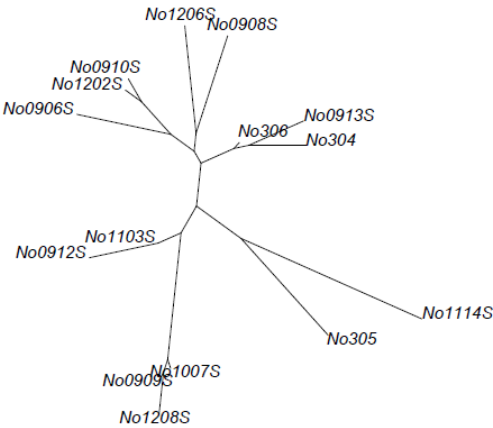


图 13-6

上面几行代码是用最简单的方法来构建一棵系统发育树并将结果可视化。但是在实际操作中，每一步都需要考虑各种模型和参数的影响，譬如计算遗传距离用的替代模型有 JC69、K80 等，构建系统发育树除了用邻接法（NJ），还可以用最小进化法（fastME）。这些问题都是分子系统学中涉及

的技术问题，详细可以参考 APE 的使用手册，此处就不展开讨论。在 R 语言中只需要调整几个命令或者参数，即可实现各种方式构建系统发育树。

### 3. 自展法（Bootstrap）与模拟数据生成（Datagen/Simulation）

构建分子系统发育树作为统计学推断的一个应用，一般还需要结论中给出一个统计学上的支持率，这时候我们一般会用自展法来计算。具体思路是将比对序列的碱基作为一个初始的样本库，接着从库中重复抽取一定数量的碱基（通常是序列的长度），根据重复抽样的序列构建系统发育树，重复此过程  $N$  次（一般为 500 次或 1000 次），并计算系统树上每个内节点出现的次数。这听起来很复杂，但 ape 包里只需要用 boot.phylo 函数即可执行自展，如图 13-7 所示。

```
1 f = function(x){nj(dist.dna(x))}
2 N=1000 #设置自展次数
3 BS <- boot.phylo(tree, woodmouse, f, N) #执行自展
4 plot(tree)
5 nodelabels(round(BS/N*100,1), adj=c(1,-0.15), frame="none") #将计算的支持率列在节点附近，一目了然
```

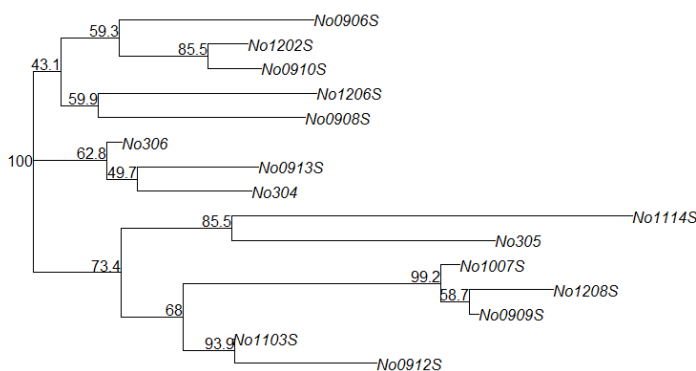


图 13-7

到这里，构建一个基因的系统发育树的基本方法就介绍完成了。当然，单纯用 ape 包来构建系统发育树似乎并不是一种很常规的方法，毕竟邻接法或者最小进化法其实已经被最大似然法（ML）和最大简约法（MP），甚至贝叶斯算法（Bayes）所替代。在 R 中也有其他的 R 包可以执行这些目前更流行的算法，如 phangorn 等。但 ape 包是系统发育分析的一个鼻祖，很多 R 包都是基于 ape 包设计的方法和类进行操作的，掌握好 ape 包的操作是入门的基础。

## 13.2.5 读懂你的对象

在 R 中使用各种 R 包时，我们往往会碰到各种各样的对象，其中保存着我们要操作的数据。从

实用的角度来看，我们只需要查阅帮助文档即可知道怎样处理这些对象。但是从一个学习者的角度来看，这些对象其实跟函数一样，是一个个“黑盒子”，尤其是当刚接触一套 R 包的时候，我们感觉就是在盲人摸象，不确定这种数据处理方法是否完全适合自己手上的那套数据。

R 软件中的每份数据都以一定“类”存在对象中，譬如：

```
1 a=c(1:10)
2 class(a)
# [1] "integer"
3 a
# [1] 1 2 3 4 5 6 7 8 9 10
```

在上面的代码中，数列 1~10 保存在对象 a 中，属于 integer 类，同时我们可以直观地访问对象 a 中的每个元素。但其中也有一些复杂的对象，如

```
1 data(woodmouse)
2 class(woodmouse)
# [1] "DNABin"
3 woodmouse
# 15 DNA sequences in binary format stored in a matrix.
# All sequences of same length: 965
# Labels: No305 No304 No306 No0906S No0908S No0909S ...
# Base composition:
#   a   c   g   t
# 0.307 0.261 0.126 0.306
```

对象 woodmouse 在前文中提到过，它是序列数据，属于 DNABin 类，直接访问 woodmouse 对象只能获得一些简略的统计信息。不过 woodmouse 中有一些是我们实在不熟悉的对象，如

```
1 tree <- nj(dist.dna(woodmouse))
2 class(tree)
# [1] "phylo"
3 tree
# Phylogenetic tree with 15 tips and 13 internal nodes.
# Tip labels:
#   No305, No304, No306, No0906S, No0908S, No0909S, ...
# Unrooted; includes branch lengths.
```

tree 这个对象为 phylo 类，直接访问此对象时，其实是不容易知道它的作用。当面对这些比较复杂的对象时，除利用 class、mode 一类的函数外，其实还可以利用 unclass 函数来访问：

```
1 unclass(tree)
# $edge
#      [,1] [,2]
# [1,]  16  22
# [2,]  22  23
# [3,]  23  25
# [4,]  25   8
# [5,]  25  10
```

```
# [6,] 23 27
# [7,] 27 28
# [8,] 28 6
# [9,] 28 15
# [10,] 27 11
# [11,] 22 26
# [12,] 26 1
# [13,] 26 12
# [14,] 16 20
# [15,] 20 21
# [16,] 21 2
# [17,] 21 9
# [18,] 20 3
# [19,] 16 17
# [20,] 17 18
# [21,] 18 5
# [22,] 18 14
# [23,] 17 19
# [24,] 19 24
# [25,] 24 7
# [26,] 24 13
# [27,] 19 4
#
# $edge.length
# [1] 2.051412e-03 1.472061e-03 1.192033e-03 3.339227e-03
-3.160774e-05
# [6] 6.045839e-03 6.610200e-04 2.302400e-04 1.972407e-03
4.403033e-04
# [11] 2.611039e-03 6.187693e-03 9.407399e-03 1.714771e-03
7.255703e-04
# [16] 2.690651e-03 2.834267e-03 3.733003e-04 6.515749e-04
8.698575e-04
# [21] 4.853986e-03 5.125078e-03 1.368619e-03 2.078914e-03
1.252824e-03
# [26] 9.483056e-04 4.562027e-03
#
# $tip.label
# [1] "No305" "No304" "No306" "No0906S" "No0908S" "No0909S" "No0910S"
# [8] "No0912S" "No0913S" "No1103S" "No1007S" "No1114S" "No1202S" "No1206S"
# [15] "No1208S"
#
# $Nnode
# [1] 13
#
```

```
# attr(,"order")
# [1] "cladewise"
```

从以上代码中我们可以很清楚且有条理地看到对象 `tree` 包含的所有信息，其中包括了 `$edge`、`$edge.length`、`$tip.label` 和 `$Nnode` 这 4 个元素。也可以用 `tree$edge` 这种方式直接访问到这些内容。`attribute` 函数也能反馈一个对象内涵的标签（`name`）和其他相关信息：

```
1 attributes(tree)
# $names
# [1] "edge"      "edge.length"  "tip.label"    "Nnode"
#
# $class
# [1] "phylo"
#
# $order
# [1] "cladewise"
```

其实，再复杂的对象，一般都是用列表（`list`）的形式进行包装的，所以一般可以用“对象”+“\$”+“元素名”的形式来访问数据。在 Rstudio 命令行状态下，输入“对象”+“\$”，然后按 Tab 键会有元素名提示。当然，有些复杂对象也有例外，譬如 `woodmouse`，在访问的时候返回的是“15 DNA sequences in binary format stored in a matrix.”。既然存储在一个数组（`matrix`）上，自然就不能用列表的方式（“\$”+Tab 键）访问里面的元素，而是需要按访问数组的形式来访问了。

### 13.2.6 修改工具包中的函数以适应新情况

使用 `read.GenBank` 函数虽然可以快速地获得大量给定基因号的序列，但如果给定的基因号有问题，则容易发生报错，结果往往是一条序列都得不到。就以小林姬鼠的例子来说，在 R Documentation 的 `woodmouse` 描述中，提到文章中研究的数据可以通过 GenBank 获取，基因号是 `AJ511877~AJ511987`，于是我们信心满满地开始下载数据：

```
1 ref = paste("AJ",c(seq(511877, 511986))), sep="")
2 read.GenBank(ref)
# Error in FI[i]:LA[i] : NA/NaN argument
```

什么？程序报错？是代码没写好还是网线没接好？抑或是其他原因？

类似这种“别人的程序很厉害，但是用在自己的项目中却报错”的情况其实很常见，尤其像 R 这种免费平台中的软件包，其作者不一定会像一些严谨的商业软件作者，通过充分的调试再发布程序。那么，如果碰到这种运行程序时报错怎么办？别慌，把“挖掘机”拆下来修理！

之前要下载的数据太多了，出错了不好找到故障点，下面先下载 10 个序列数据试一试：

```
1 read.GenBank(ref[1:10])
# 10 DNA sequences in binary format stored in a list.
#
# Mean sequence length: 853.6
# Shortest sequence: 157
```

```
# Longest sequence: 965
#
# Labels: AJ511877 AJ511878 AJ511879 AJ511880 AJ511881 AJ511882 ...
#
# Base composition:
#   a   c   g   t
# 0.307 0.262 0.127 0.305
```

从上面的代码中可以发现，前面的 10 个序列没问题，从而可以排除代码出错或者网络故障了。估计是这里面的基因号被掺进了“沙子”，导致函数不能输出正常的结果。部分数据能成功下载说明下载功能没有问题，我们只要排除 ref 数据集中有问题的基因号，就能得到目的基因序列了。要一个个地排除似乎又太耗时间。当然，我们可以利用 `read.GenBank` 函数为内核构建一个新循环，然后利用 `tryCatch` 函数绕过有问题的基因号，这是解决问题的一种思路。

但在 R 中，R 包里的函数都是开源的，所以我们还有另一种解决方法——直接检查并修改 `read.GenBank` 函数，以绕过有问题基因号。

简单地说，一般 R 包中除了样本数据，剩下的基本就是函数了，而这些内容都是开源的，我们可以根据自己的需要进行修改。读取一个函数的方法很简单，在命令行上输入函数名称就可以了。当然，这需要我们有一些 R 编程基础，这样可以比较容易理解这个工具是怎样运作的：

```
l read.GenBank
# function (access.nb, seq.names = access.nb, species.names = TRUE,
# gene.names = FALSE, as.character = FALSE)
# #定义函数的基本格式：function(#参数#){表达式}。
# {
#   N <- length(access.nb)
#   nrequest <- N%/%400 + as.logical(N%400)
#   X <- character(0)
#   for (i in 1:nrequest) {
#     a <- (i - 1) * 400 + 1
#     b <- 400 * i
#     if (i == nrequest)
#       b <- N
#     URL <- paste("http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=",
#       paste(access.nb[a:b], collapse = ","), "&rettype=gb&retmode=text",
#       sep = "") #这一步是把多个基因号编辑成一条 url，然后让下一行的 scan 函数访问 ncbi 读取序列数据。我们可以把这条命令单独拿出来，把 access.nb[a:b]换成 ref，执行后得到的 URL 可以直接放到浏览器的地址栏上，ncbi 会返回一个 genbank 格式的文件，文件内包含了所有可以获取的基因序列。有些是空值的基因号，但不会影响这一步的正常进行。
#     X <- c(X, scan(file = URL, what = "", sep = "\n", quiet = TRUE))
#   }
#   FI <- grep("^ORIGIN", X) + 1
```



```

#   LA <- which(X == "//") - 1
#   obj <- vector("list", N)
#   for (i in 1:N) {
#       tmp <- gsub("[[:digit:]]", "", X[FI[i]:LA[i]]) ### "FI[i]:LA[i]"
#   }
#   names(obj) <- seq.names
#   if (!as.character)
#       obj <- as.DNABin(obj)
#   if (species.names) {
#       tmp <- character(N)
#       sp <- grep("ORGANISM", X)
#       for (i in 1:N) tmp[i] <- unlist(strsplit(X[sp[i]], " +ORGANISM
+"))[2]
#       attr(obj, "species") <- gsub(" ", "_", tmp)
#   }
#   if (gene.names) {
#       tmp <- character(N)
#       sp <- grep(" +gene +", X)
#       for (i in 1:N) tmp[i] <- unlist(strsplit(X[sp[i + 1L]],
#         " +/gene=\\\""))[2]
#       attr(obj, "gene") <- gsub("\\\"$", "", tmp)
#   }
#   obj
# }
# <environment: namespace:ape>

```

了解了 read.GenBank 函数的运作方式以及出现问题的原因，接着处理问题的思路就清晰了。我们可以在这个函数基础上进行适度修改，以顺利获取数据。

```

1 read.GenBank_test= #最好不要把原来的函数覆盖，在原来的函数名后边加个后缀就可以了。
2 function (access.nb, seq.names = access.nb, species.names = TRUE,
3   gene.names = FALSE, as.character = FALSE)
4 {
5   N <- length(access.nb)
6   nrequest <- N%%400 + as.logical(N%%400)
7   X <- character(0)
8   for (i in 1:nrequest) {
9     a <- (i - 1) * 400 + 1
10    b <- 400 * i
11    if (i == nrequest)
12      b <- N

```

```

13      URL <- paste("http://eutils.ncbi.nlm.nih.gov/entrez/eutils/
efetch.fcgi?db=nucleotide&id=",
                    paste(access.nb[a:b], collapse = ","), "&rettype=gb&retmode =text",
                    sep = "")
14      X <- c(X, scan(file = URL, what = "", sep = "\n", quiet = TRUE))
15  }
16  FI <- grep("^ORIGIN      ", X) + 1
17  LA <- which(X == "//") - 1
### 下面几行用于重新计算 N 值，seq.names 也需要重新赋值，只需要考虑 ncbi 返回的序列即可。
18      N <- length(grep("LOCUS", X))
19      seq.names = c()
20      for (i in 1:N){
21          seq.names <- cbind(seq.names, strsplit(X[grep("LOCUS",
X)], "      ")[[i]][2])
22      }
###
obj <- vector("list", N)
for (i in 1:N) {
    tmp <- gsub("[[:digit:]]", "", X[FI[i]:LA[i]])
    obj[[i]] <- unlist(strsplit(tmp, NULL))
}
names(obj) <- seq.names
if (!as.character)
    obj <- as.DNABin(obj)
if (species.names) {
    tmp <- character(N)
    sp <- grep("ORGANISM", X)
    for (i in 1:N) tmp[i] <- unlist(strsplit(X[sp[i]], " +ORGANISM +"))[2]
    attr(obj, "species") <- gsub(" ", "_", tmp)
}
if (gene.names) {
    tmp <- character(N)
    sp <- grep(" +gene +", X)
    for (i in 1:N) tmp[i] <- unlist(strsplit(X[sp[i + 1L]],
        " +/gene=\"")[2]
    attr(obj, "gene") <- gsub("\"$", "", tmp)
}
obj
}

```

根据原有的函数进行修改的好处在于函数使用起来比较方便，不需要构建全新的循环函数，节省了解释参数这些工作，代码维护起来也非常简单。另外，多读牛人写的函数，能快速地提高自身的编程能力。

# 第 14 章

## 产品化： 关于内存、速度和自动化

本章是本书的结尾，会重点介绍 R 脚本的自动化及内存、速度等新手比较关注的问题，因为新手总是纠结于工具的缺点。下面主要介绍这些工具的“缺点”和修复方法。

### 14.1 不同终端调用、自动化执行 R 脚本及参数传递

一个分析脚本写完了，重点是最后能批量产出。也就是说，我们每次执行一个已经完成的脚本，脚本就能根据数据产出结果。脚本应该以最大的容错能力产出最终的结果。R 脚本文件的后缀是.R，比如下面是一个缩写的用于对城市排名的脚本“cityrank.R”，它的内容如下：

```
# 读取数据
1 hpi <- read.csv(file="H:/data/hpi.csv", header = T, sep = ",", row.names
= 2)
2 data <- dplyr::select(hpi, -Country, -HappyPlanetIndex)
# 主成分分析方差提取
3 datascale <- scale(data)
4 library(psych)
5 rc <- principal(datascale, nfactors = length(datascale[,]),
rotate = "varimax", scores = TRUE)
6 comp <- as.data.frame(unclass(rc$scores))
7 p <- print(rc)
8 pcweight <- data.frame(p$Vaccounted)
# 主成分加权法综合排名
9 result <- t(as.matrix(comp)) * unlist(pcweight["Proportion Var",])
10 result <- t(result)
11 result <- apply(result, 1, sum)
12 result <- (result - min(result))/(max(result) - min(result))
```

```
13 result <- result*100
14 princompresult <- cbind(hpi, comp, result)
15 write.csv(princompresult, "princompresult.csv")
# 文件写在了当前目录下
```

R 脚本具有以下能力才能自动化执行。

第一，对于自动化的脚本，要求至少应该在无人工参与的情况下能正常执行。如果一个脚本经常需要人工参与调整参数、重置个别步骤，显然是不具有自动化的水平，需要进一步去掉人工参与的内容。

第二，要有一定的容错能力，比如在上面的代码中，第 2 行代码“data <- dplyr::select(hpi, -Country, -HappyPlanetIndex)”，使用了列的名称进行剔除列，当然也可以写成“data <- hpi[, -c(1,5)]”，但是数据结构一旦被改变，比如第一列不是 Country 列了，这样的剔除就失效了，所以要避免使用这种不能容错的语句。

第三，脚本可以被拆分。将复杂的脚本拆成几个具有独立功能的小脚本，这样做首先降低了脚本的复杂度，另外，脚本被拆分后，脚本的执行先后顺序和时间安排更加灵活了。

当 R 脚本达到自动化执行的要求后，不用人工一句句地点击代码执行，可以使用 source 函数批量执行。

- source

```
1 source("H:/data/cityrank.R")
```

使用 source 函数可以执行一个完整的 R 脚本，参数只需要指定 R 脚本的完整地址就可以了。这样 R 程序就会直接执行脚本 cityrank.R。

在 Windows 环境下，如果你的 R 环境变量已经配置完成，那么在 Windows 桌面的“开始”菜单的搜索栏中输入“cmd”，再按 Enter 键，就会启动 Windows 的 cmd 窗口。在其中输入大写的“R”，就能正常启动 R 脚本，然后输入“q()”退出窗口，如果没有启动 R 脚本，则说明你的环境变量还没有配置成功。

- 在终端直接调用 R 脚本

```
1 Rscript H:/data/cityrank.R
2 /usr/local/bin/Rscript /root/Documents/code/2015-06-29/cityrank.R
```

在上面的代码中，第 1 行代码可以在 Windows 中启动 cmd 窗口，然后直接调用 R 脚本并执行结果。请记住，在 R 脚本中尽量不要使用中文，特别是名称、路径等，因为一旦出现编码不兼容的情况，就会出现乱码，不能正常执行脚本。第 2 行代码是在 Linux 环境下终端调取 R 脚本的方法，前半部指定 R 程序的地址，后半部分指定 R 脚本的地址，这样就可以在终端调用了。

有时候我们需要按照一定时间开启和关闭一个程序，比如定期抓取网页数据，但是如果做循环则需要时时刻刻开启 R 程序，为了节省计算机内存，我们需要定时执行开启 R—执行脚本—关闭 R 程序这样一个循环，这时可以用 cron，用于设置周期性被执行的指令。

- 在 Linux 环境下安装 crontabs

```
1 yum -y install vixie-cron
2 yum -y install crontabs
```

vixie-cron 包是 cron 的主程序；crontabs 包是用来安装、卸装或列举用来驱动 cron 守护进程的表格的程序，两个软件包均使用 yum 安装即可。

- 添加 crontab 任务

```
1 vim /etc/crontab
```

在 Linux 系统中，上面的代码使用 vim 命令打开 crontab 文件，并把下面的内容加在 crontab 的最后，然后按 W+Q 组合键保存。

- 添加 crontab 任务

```
1 */2 * * * * Rscript root/Documents/code/cityrank.R
```

上面代码中的\*/2 表示每隔两分钟运行一次 R 脚本，运行的 R 脚本 cityrank.R 放在 root/Documents/code 路径下。crontab 任务前面的 5 个 “\*” 符号分别代表分钟、小时、日、月、周，例如要在每周六的 12 点 13 分运行 cityrank.R 脚本，则需要在 crontab 文件中添加如下代码：

```
1 13 12 * * 6 /usr/local/bin/Rscript /root/Documents/code/2015-06-29/cityrank.R
```

终端开启或关闭 crontabs 的代码如下：

```
1 service crond start //启动服务
2 service crond stop //关闭服务
3 service crond restart //重启服务
4 service crond reload //重新载入配置
5 service crond status //查看 crontab 服务状态
```

测试后不要忘记关闭或者注释掉自己的任务，不然它会定时运行。

但是，有时候任务不是只运行一个 R 脚本，可能会牵涉 Java 任务、Python 任务、数据库迁移任务，这时就需要使用一个统一的调度工具来完成。我一直使用 kettle 作为统一调度工具。有时候使用其他语言调用 R 语言脚本比较麻烦，但是调用 shell 脚本很容易，于是我想到了用其他语言调用 shell 脚本，然后用 shell 脚本调用 R 脚本。可以先创建一个.sh 的文件，用记事本程序在里面添加以下代码：

```
1 /usr/local/bin/Rscript /root/Documents/code/2015-06-29/cityrank.R
```

然后让其他语言调用这个 shell 脚本就可以完成统一调度。

介绍完调用和自动化执行脚本，下面要介绍一下参数传递了，也就是我们在调用 R 脚本的时候能不能传递给 R 脚本几个参数？首先要创建一个接受参数的 R 脚本，取名为 argtest.R，它的内容如下：

```
1 Args <- commandArgs()
2 cat("Args[1]=",Args[1],"\n")
3 cat("Args[2]=",Args[2],"\n")
4 cat("Args[3]=",Args[3],"\n")
5 cat("Args[4]=",Args[4],"\n")
6 cat("Args[5]=",Args[5],"\n")
7 cat("Args[6]=",Args[6],"\n")
8 cat("Args[7]=",Args[7],"\n")
9 temp <- Args[6]
```

```
10 writ.csv(temp, "/data/argtest.csv")
```

在上面的代码中，第 1 行代码使用 `commandArgs` 函数接受所有传过来的参数，并全部存放在 `Args` 中。`Args` 中的前 5 个参数是其本身的参数，从第 6 个起是我们传入的参数。第 2~8 行代码为打印每一个参数。第 9 行代码将第 6 个参数，也就是我们传入的第一个参数赋值给 `temp`。第 10 行代码将结果写到 `argtest.csv` 文件中。这样就可以在终端调用 R 脚本并传递参数了。

- R 脚本传递参数

```
Rscript /root/Documents/argtest.R 这个第一个参数 这是第二个参数
```

此方法在 Linux 系统下执行，参数之间用空格隔开即可，这样就不仅实现了调用 R 脚本的同时，还实现了传递参数，接收到传递的参数之后又可以调用其他脚本对参数进行分析挖掘了。

通过以上方法让 R 程序变得更开放了，虽然比较曲折，但是这让 R 程序的适用范围更加广泛，实现了和其他程序的统一执行和相互沟通。有了可以传递参数的方式，我们使用 `shell` 或其他统一调度的工具如 `kettle` 等调取 R 脚本就易如反掌了。

## 14.2 与速度、内存、并行相关的程序优化

计算速度和计算机内存往往是计算大量数据的难点，如果计算机内存不够，则很多算法很难实现。虽然最简单的办法就是为机器增加内存，但是，在平时处理数据时我们也要注意一些细节以加强内存管理。

R 会自动回收内存，但是回收的速度太慢，无法及时解决问题，我们可以通过及时移除一些比较大的无用对象，减少对内存的占用。

- 清除对象，回收内存

```
1 rm(a, b)
2 gc()
3 rm(list = ls())
4 gc()
```

在上面的代码中，第 1 行代码移除对象 `a`、`b`。第 2 行代码回收对象所占的内存。第 3 行代码使用 `ls` 函数获得内存中的对象名称，然后赋值给 `rm` 函数的参数 `list`，这样就移除了内存总所有的对象。第 4 行代码再次回收内存。

上述方式比较烦琐，我们在编写程序时可以及时将一些仅用于临时几个步骤的对象统一命名，比如将其命名为“`tmp`”，直到此对象完成它的历史任务后，再将其他临时对象命名为“`tmp`”，这样内存中一直只有一个“`tmp`”，既减少了对对象，自动节省了内存，也便于移除对象，例如：

```
1 tmp <- 1:5
2 a <- tmp + 3
3 .....
4 tmp <- 1:3
5 a <- a + tmp
```

但是，有些时候我们还是逃不过内存不够的情况，那么只能将数据先分块整理，看一看代码中的哪一部分最耗内存，然后修改代码以减少占用的内存。上面的方法还是雕虫小技，我们还是应该寻找更加节省内存的方法，比如过于稀疏的矩阵要使用稀疏矩阵存储，而且要寻找基于这类节省内存的方式的算法。

其实上面的方法只能暂缓燃眉之急，还是增加内存更方便。即使过了内存这一关，代码的运行速度问题也会让人愁白头的。下面就根据我的“二十多年的经验”聊一聊运行速度方面的问题吧。首先代码的速度优化应该分为 4 个层次：语言层次、函数层次、设备层次和思路层次。

语言层次优化的重点在语言的理解上，比如前面就已经讲了 R 语言在循环方面具有劣势是因为它是一个向量化的分析过程。也就是说在使用 R 语言时，我们要尽最大努力将循环分析过程向量化或矩阵化，这样可以节省大量的程序运行时间。但是根据我多年的经验，有时候需要故意使用一些循环，比如在做文本分析时，虽然我们每天搜集到的文本数据比较少，但是仍然可能会面对大量的历史数据，即使使用向量化 R 脚本也需要一两天才能完成。在第一次分析这些大量的历史数据时，程序难免会被一些特殊的情况搞得脚本崩溃，向量化的代码一旦崩溃，内存中是不会保留已经分析完成的结果的，只能从头再来。这样一味地固守向量化就可能会浪费我们很多时间（多次从头再来），所以，我们需要循环和向量化相结合的方法。只需要将数据切成较小的 10 份，然后循环执行向量化的脚本即可，这样即使其中某一份数据报错，我们找到错误后，只需要从该份数据开始再次循环就可以了，从而为我们节省大量的时间。

函数层次的优化是大家经常碰到的，在第 2 章中讲解了很多这方面的情况，比如常见的函数都没有 data.table 包中的快，如 rbind 函数的速度比不上 rbindlist 函数，用于匹配的 %in% 函数的速度比不上 %chin% 函数，stri\_extract\_all\_regex 函数的速度远远快于 str\_extract\_all 函数。所以，当你碰到自己用的函数成了脚本速度的瓶颈时，寻找更加快速的函数也是一种速度优化的方法。

设备层次的优化主要指并行运算和集群，这里主要介绍一下 R 语言中的并行计算。并行计算只有在数据量比较大时才具有优势，因为启动一下程序都需要花费可观的时间。下面演示一下在 Linux 系统下并行计算的方法，即进行并行分词。

- 并行分词

```
1 library(Rwordseg)
2 library(parallel)
3 cl <- makeCluster(getOption("cl.cores", 10))
4 clusterEvalQ(cl, library(Rwordseg))
5 system.time(x <- parLapply(cl, sentence, function(x) { x <- segmentCN(x) }))
6 stopCluster(cl)
```

在上面的代码中，第 3 行代码声明要开启的线程数。第 4 行代码使用 clusterEvalQ 函数将要用到的包绑定到各个线程上。第 5 行代码使用 parLapply 函数进行多线程分词，第 1 个参数为声明线程的变量，第 2 个参数指定用于分词的句子向量，第 3 个参数用于指明函数。这里使用了一个匿名函数，调取了 Rwordseg 包中的 segmentCN 函数。当面对大量数据时，使用这种方式往往快速很多。

为什么不用分布式的集群，如 Spark 等？基本上有两个原因，其一，这类大型的集群启动一次

也需要不少时间，数据挖掘流程一般在数据整理和首次建模时比较耗时，因为使用的历史数据跨度比较大，比如预测客户流失可能性，我可能使用过去 2~3 年的历史数据，而模型建立后，我只需要预测新数据就可以了”，至少和第一次建模相比要小很多，这样 R 还是秒级完成的，所以，如果动用那些 Spark 工具，也就第一次会用到，后面就没必要用了，如果后面也用 Spark 或者 Hadoop 等，启动一次都要超过了秒级，反而拖慢了程序。其二，这类工具对算法和 R 的支持不是很好，你花了很多时间搞定了集群，R 的脚本和你在 Spark 上调用的 R 的脚本是两个完全不同的脚本，很多包都不支持，你不得不花费大量的时间“造轮子”。

思路层次的优化是最容易被人忽略的，如果存在 a 和 b 两个处理步骤，b 可以在先，a 也可以先，那么你是先执行 a，还是先执行 b？不同的安排会对脚本的速度产生影响吗？答案是肯定的，因为每一步的函数在应对大量数据时，效率都不一样。我们的策略是将速度更快，同时能更显著地减少下一步的数据输入的处理步骤放在前面。比如对于去除停用词和计算词频这两个处理步骤，当面对巨量数据时，你就要考虑一下应该先完成哪个步骤。

综上所述，无论 R 脚本还是 Python 脚本的速度优化，基本可以遵照上述方式进行优化，请记住：天下武功，唯快不破！